Utilising Reinforcement Learning for the Diversified Top-k Clique Search Problem

MSc Thesis *(Afstudeerscriptie)*

written by

Jesse van Remmerden
(born 09, 07, 1995 in Leeuwarden, the Netherlands)

under the supervision of **dr. S. Wang** with **dr. ir. D. Thierens** as Second Examiner in partial fulfillment of the requirements for the degree of

Master of Science (MSc.) in Artificial Intelligence

at *Utrecht University*.

Date of the public defense: *23, 06, 2022*

Members of the Thesis Committee:

Utrecht University

Abstract

The diversified top-k clique search problem (DTKC) problem is a diversity graph problem in which the goal is to find a clique set of k cliques that cover the most nodes in a graph. DTKC is a combinatorial optimisation problem and can be seen as a combination of the maximum clique problem and maximal clique enumeration. In recent years, a new research field arose that research if reinforcement learning can be used for combinatorial optimisation problems. However, no reinforcement learning algorithm exists for DTKC or any other diversity graph problem. Therefore, we propose Deep Clique Comparison Agent (DCCA), which utilises PPO, Graph Isomorphic Networks and the *encode-process-decode* paradigm to compose an optimal clique set. We tested DCCA for DTKC and the diversified top-k weighted clique search problem (DTKWC). Our results showed that DCCA could outperform previous methods for DTKC, but only on higher values of k, such as if $k = 50$. However, we only saw this occur on simpler graphs and DCCA performed significantly worse on the other problem, DTKWC. Due to the novelty of DCCA, we believe that future research can significantly improve our results.

Contents

Chapter 1

Introduction

Reinforcement learning (RL) is one of the three machine learning paradigms. The goal of RL is to get an agent to behave in such a manner that it will maximise its reward based on the environment and the current state of that environment. This definition sounds complex, but it can be easily explained by using a chess game as an example. The goal of chess is to capture your opponent's king while making sure your opponent does not capture your king. When it is your turn, you need to decide what action brings you closer to that goal. This action can be any available move for that current board state, even sacrificing one of your pieces, so long if that action results in you winning. This way of decision-making is what an RL agent should learn, thus not only the best move given the current state but also what it needs to do to go the best state, which is when it captures its opponent's king.

In the recent years, there has been a rise in interest in RL. There are various reasons for this, like advancements in self-driving cars (Kiran et al., 2021) and when AlphaGo (Silver et al., 2016) defeated the number one Go player in the world. This defeat surprised a lot of machine learning researchers because Go is one of the most complex games, and they were not expecting such a program to be possible at that time. This victory showed the power of RL for complex problems. Since then, there has been a lot of research about utilising RL on another set of problems, namely, combinatorial optimisation problems.

A combinatorial optimisation (CO) problem is a problem that has a finite set of solutions, of which only one is the most optimal. At first sight, this sounds not difficult to achieve; however, two essential traits of CO problems make this process not only difficult but almost impossible to achieve. Firstly, to find the optimal solution, all the possible have to be checked to ensure that the optimal solution is the optimal solution. If the set of possible solutions is not too big, then this would not be a problem. Unfortunately, for most CO problems, the number of possible solutions can quickly grow larger than the number of stars in the observable universe. For example, a travelling salesman problem (TSP)[1] instance with 24,978 cities would have approximately 1.529×10^{138446} possible solution; in comparison, the number of possible moves in Go is 10^{130} (Licht-

[1] See section 2.2 for an explanation of the problem

enstein and Sipser, 1980), and the estimated number of atoms in the whole observable is between 10^{78} and 10^{82} (Villanueva, 2018). A solution for the TSP instance, with 24,978 cities, was found, which would take an Intel Xeon 2.8 GHz processor around 84.8 years to compute (Applegate et al., 2010).

The execution time needed for finding an optimal solution is almost always an issue. For instance, if a navigation system takes too long to find the best route, it would render it completely useless. Therefore heuristic algorithms are mainly used for CO problems. A heuristic algorithm does not guarantee that it will find the optimal solution but rather that it finds a good enough result in a reasonable time.

Using reinforcement learning for finding solutions for CO problems is a new but emerging research field. One of the reasons is that many CO problems can easily be formulated as a Markov Decision Process - especially the reward function - because it is always evident when one solution is better than another (Mazyavkina et al., 2021). Another significant reason is the lack of suitable labelled training data. It is easy to create an instance for most CO problems, but labelling the correct answer is expensive operation (Cappart et al., 2021). This labelling cost is why supervised learning is less used on CO problems.

This thesis will propose a novel reinforcement learning approach for the diversified top-k clique search problem, which can be extended to other diversity graph problems. This research aims to determine if reinforcement learning will improve the previously established results of the diversified top-k clique search problem on either the execution time or final score, through a new reinforcement learning algorithm called the Deep Clique Comparison Agent (DCCA).

We start by giving relevant background information, which is essential for understanding our research question. After that, we state the main and sub research questions for this thesis. The next chapter shall go more in-depth on the topics discussed in our background section and focuses on relevant information for our algorithm.

Our methodology chapter explains how we designed our algorithm and show our argumentation for these design choices. Next, we will discuss our experimental setup, in which we state how we conduct our experiments and how we will compare DCCA to other non-RL methods, which act as our baselines. Lastly, we will show the results of our experiments and explain them in our discussion chapter.

1.1 Background

This section shows essential background information. The first subsection discusses the essential parts of graph theory, such as the notation we will use for graphs and important definitions, such as the definition of a clique. After that, we show the problem statement of the diversified top-k clique search problem (DTKC), and we discuss related diversity graph problems. However, we will not discuss previous approaches for it, those we discuss in our literature review. Our next section gives background information about reinforcement learning (RL) and states essential definitions of it. In it, we will also show some RL algorithms; however, this is only done such that we can give a better background to important definitions. The RL algorithms, we considered for our approach will be explained in the literature review. Lastly, we explain how graph neural

networks function because we believe this is the best method to encode graphs for our research.

1.1.1 Graph Theory

Graph theory is the study of graphs and is said to be introduced by Euler in his paper about the seven bridges of Königsberg (Euler, 1741). Consequently, people have used graph theory to explain interactions in various applications, such as molecular biology (Huber et al., 2007), social network analysis (Otte and Rousseau, 2002) and the spread of COVID-19 (Alguliyev et al., 2021). This section will explain the basics of graph theory necessary to understand the diversified top-k clique search problem (DTKC).

The simplest definition of a graph $\mathcal{G} = (V, E)$ consists of two sets: a set of nodes (V) and a set of edges (E). A node expresses an object within a graph, while an edge between two nodes defines a relationship between the two. Both an edge and node can contain attributes. These attributes can be anything. For example, which group a node belongs to or the weight of an edge. An edge can either be undirected or directed. An undirected edge can be traversed from either node. Such an edge could be used to define a friendship relationship between two people or a two-way street between two locations. If an entity is at a node with directed edges, that entity can only move from that node to neighbouring nodes if an edge is directed to that neighbouring node.

An edge in the set of edges E is a tuple with two nodes (u, v), with $u \neq v$, $y \in V$ and $v \in V$. If an edge is undirected, then $(u, v) = (v, u)$, but if an edge is directed then $(u, v) \neq (v, u)$, because the edge points from node u to node v. This definition of an edge allows us to define more complex functions that describe a node's property. Two important properties are finding all the neighbouring nodes and the degree of a node. These properties are essential in the later definitions of DTKC.

Definition 1.1.1. *Neighbourhood and Degree* - The neighbourhood of a node v, graph $\mathcal{G} = (V, E)$, is the set $N(v, \mathcal{G}) = \{u \in V | (v, u) \in E\}$. This set contains all the nodes connected to v. The degree of node v is $d(v, \mathcal{G}) = |N(v, \mathcal{G})|$.

The degree and neighbourhood are essential because we need to find maximal cliques in a graph (see definition 1.1.2). In essence, the degree helps us find the best-connected node, and the neighbourhood set allows us to find the maximal clique from this node. However, finding a maximal clique can be difficult because it is an NP-Complete problem (Karp, 1972). This complexity means that there is currently no algorithm that can easily find a maximal clique, but a clique can easily be verified as a maximal clique.

Definition 1.1.2. *Maximal Clique* - A clique C, in a graph $\mathcal{G} = (V, E)$ is a set of nodes $C \subseteq V$, such that all nodes are connected to each other. This clique C is then maximal if there exists no other clique C' for which $C \subseteq C'$.

The definition of a clique is strict in that all the nodes in the clique need to be connected. If needed, it is possible to loosen this definition to become a s-plex (see definition 1.1.3). A s-plex (Seidman and Foster, 1978) is similar to a clique, except each node does not need to be connected to all other nodes. The complexity of the s-plex problem is also NP-complete (Balasundaram et al., 2011).

Definition 1.1.3. *s-plex* - A subgraph $P \subseteq \mathcal{G}(V, E)$ is a s-plex if the following holds: $\min_{v \in V(P)} d(v, P) \geq |V(P)| - s$.

Cliques and s-plex are examples of subgraphs, which algorithms can find through constraints. However, it is also possible to search directly for subgraphs in a given graph G, which are isomorphic (see definition 1.1.4) to a queried graph. Finding these subgraphs is called the subgraph isomorphism problem, which is again NP-Complete (Cook, 1971).

Definition 1.1.4. *Graph Isomorphism* - Graph G and graph H are isomorphic $G \simeq H$ to each other, if there exist a function: $f : G \mapsto H$, for all $u, v \in V(G)$, $(u, v) \in E(G) \Leftrightarrow (f(u), f(v)) \in E(H)$

1.1.2 Diversified Top-k Clique Search

The diversified top-k clique search problem (DTKC) is formulated in the paper "Diversified Top-k Clique Search" (Yuan et al., 2015). The goal of DTKC is to find k cliques, such that most nodes in the graph are covered. Yuan et al. (2015) describe how DTKC combines two other combinatorial optimisation (CO) problems problems, namely, maximal clique enumeration (MCE) and max k-cover[2]. Both these problems have been studied extensively, and previous work also researched the combination of the two problems. In these methods, first, all the maximal cliques are found in the graph. Then, from those cliques, k cliques are picked which cover the most nodes (Feige, 1998; Lin et al., 2007). However, these methods do not scale to larger graphs because the number of cliques in a graph grows exponentially with the number of nodes in a graph (Eppstein et al., 2010). Yuan et al. (2015) tries to alleviate this by always keeping only k cliques in memory.

Definition 1.1.5. *Coverage* - The coverage of clique set $\mathcal{D} = \{C_1, \dots C_k\}$ is all the nodes of the cliques $C \in \mathcal{D}$.

$$\text{Cov}(\mathcal{D}) = \bigcup_{C \in \mathcal{D}} C \tag{1.1}$$

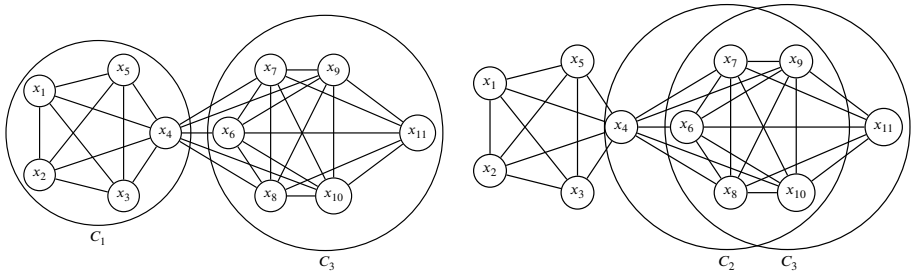
For example, the coverage of figure 1.1a would be $\text{Cov}(\{C_1, C_3\}) = \{x_1, \dots, x_{11}\}$, while the coverage of figure 1.1b would be $\text{Cov}(\{C_2, C_3\}) = \{x_4, x_6, \dots, x_{11}\}$

Problem Statement DTKC. The problem statement of DTKC states: Given a graph $\mathcal{G}$ and an integer k, the goal of DTKC is to find a set of cliques $\mathcal{D}$, such that $|\mathcal{D}| \leq k$, any $C \in \mathcal{D}$ is a clique and $|\text{Cov}(\mathcal{D})|$ is maximised.

One approach for finding a diversified top-k clique set is to find the largest k cliques in a graph. At first sight, this approach seems effective because the largest k cliques contain the most cliques in total. However, previous work shows that large cliques are likely to overlap (Wang et al., 2013). Because of this, the set of largest k cliques is likely not the most diverse clique set because a lot of cliques in the set will overlap. How this happens is shown in example 1.1.

[2]Sections 2.2.2 and 2.2.3 will discuss these problems in-depth

Example 1.1. The graph in figure 1.1 has three cliques: $C_1 = \{x_1, x_2, x_3, x_4, x_5\}$, $C_2 = \{x_4, x_6, x_7, x_8, x_9, x_{10}\}$ and $C_3 = \{x_6, x_7, x_8, x_9, x_{10}, x_{11}\}$, and we set $k = 2$. The figure shows that picking cliques C_1 and C_3 would lead to the most diverse set, even though $|C_1| < |C_2|$. This example also shows why the set of largest k cliques is not always the most diverse set.



(a) Diversified Top-2 Cliques

(b) Top-2 Maximal Cliques

Figure 1.1: There exist three cliques in this graph and although C_2 is larger, the combination of C_1 and C_3 covers the most nodes.

Besides DTKC, there also exist many other related diversity graph problems. An excellent example is the diversified top-k s-plex search problem (Wu and Yin, 2021a), which uses s-plexes instead of cliques. Another example is diversified top-k subgraph querying (DTKSQ) (Fan et al., 2013; Yang et al., 2016; Wang and Zhan, 2018). With DTKSQ, the goal is to find a set k subgraphs that is isomorphic to the queried graph. However, diversified top-k weighted clique search problem (DTKWC) is most similar to DTKC, except that the goal is not to find the clique set that maximises the coverage but that the summation of the nodes' weights in the coverage is maximised. We will use the definition of DTKWC a lot in this thesis and, therefore, we will state the complete problem statement for it:

Problem Statement DTKWC. The problem statement of DTKWC states: Given a weighted graph $\mathcal{G}$, with a weight function $w(u) \in \mathbb{Z}$ and an integer k, the goal of DTKC is to find a set of cliques $\mathcal{D}$, such that $|\mathcal{D}| \leq k$, any $C \in \mathcal{D}$ is a clique and $\sum_{u \in \mathrm{Cov}(\mathcal{D})} w(u)$ is maximised.

1.1.3 Reinforcement Learning

The introduction described that reinforcement learning (RL) aims to get an agent to learn to behave in an environment such that it maximises its cumulative reward. This section will explain essential concepts of this field, such as how to formulate a reinforcement learning problem, the difference between Model-Based and Model-Free algorithm, and the exploration-exploitation trade-off.

Two essential components in RL are the agent and the environment. The agent is the RL algorithm, which makes decisions based on the current state of the environment,

which is the place where the agent operates. An environment can be anything from a game of Mario to how a robot interacts in the real world. These examples are entirely different in terms of what their goal is and how an RL agent should behave in the environment. However, a Markov decision process (MDP) can formulise how an agent should act in each environment (Bellman, 1957). An MDP describes what kind of actions are possible, the different states of the environment, the reward function and how to get to each state. This thesis will use the MDP notation used by Mazyavkina et al. (2021).

S **- state space** $s_t \in S$ A state describes the current setting of the environment. This state is everything that the agent needs to make a decision. The state-space is the set of all the possible states in the environment. This set can both be finite, in the case of chess, or be infinite if the state contains real numbers.

A **- action space** $a_t \in A$ The action space describes all the possible actions for the agent. An action can be one or multiple values, depending on the environment. Each value in the action variable can either be continuous or discrete.

R **- Reward function** $R : S \times A \to \mathbb{R}$ The reward function maps a state and an action to a real number. The reward indicates how well the agent's action was at that state.

Transition function $T(s_{t+1}|s_t, a_t)$ The transition function dictates the transition between states through the action chosen by the agent.

Discount factor γ The discount factor γ indicates whether the agent will prefer a short-term or long term reward. If γ is close too 1, the agent will prefer the long-term reward, and if $\gamma = 0$ meaning that the agent will only opt for the short term reward.

H **- horizon** The horizon is the length of the episode. An RL task can either be episodic, which says that there is a terminal state, or continuous, which means that there is no state at which the environment will stop. Each RL solution for a CO problem will be episodic.

RL algorithms can be divided into two categories: Model-Based and Model-Free algorithms. Model-Based algorithms have access to a model of the environment or can learn this model. If an algorithm is Model-Based, it will rely on planning. This capacity to plan is possible because the algorithm knows what states are possible in the future and what actions it can take based on these states (Sutton and Barto, 2018). One of the best-known model-based algorithms is Monte Carlo tree search (MCTS) (Coulom, 2007). MCTS will decide which action to take at each state based on simulated outcomes of all the possible actions and then takes the action with the highest estimated reward. The reason it can do this is that it knows the whole model of the environment. For instance, it knows the possible actions for both itself and all the other agents in the environment. Through this knowledge, it can simulate the outcome of the environment from any given state.

Model-Free algorithms are, as the name implies, not based on any model of the environment and thus do not know the transition function. Instead, these algorithms decide which action to take based on previously earned rewards. These agents learn this through trial-and-error by interacting with the environment. An essential aspect of this is the exploration-exploitation trade-off (Sutton and Barto, 2018). This trade-off is not only applicable to RL but also to how we people learn in our life. It explains the dilemma of choosing the action, which, according to our current knowledge, leads to the best reward, or exploring new actions, which can lead to a better reward, but also at risk it can result in a lower reward.

A common way of balancing the exploration-exploitation trade-off in RL is through the ϵ-greedy strategy (Sutton and Barto, 2018). With this strategy, the RL agent will have a $1-\epsilon$ chance of exploiting the current action and an ϵ chance of exploring through picking a random action. However, this strategy is in most cases not optimal because ϵ is static, and in most environments, an RL agent would benefit the most from exploring at the start of the learning process because it lacks any knowledge about it and only should start to exploit more when the agent has enough knowledge about the environment. Modified versions of ϵ-greedy try to solve this problem. For example, annealing ϵ-greedy (Akanmu et al., 2019) starts with a high ϵ and will lower over time. Another version is adaptive ϵ-greedy (Mignon and A. Rocha, 2017), which decides to lower or higher ϵ based on the current results.

Two of the most well-known Model-Free algorithms are SARSA (Rummery and Niranjan, 1994) and Q-Learning (Watkins and Dayan, 1992), with both algorithms trying to achieve the same: learning the best action for a given state. Both algorithms learn the quality of a state-action pair $Q(S_t, A_t)$ through temporal difference (TD) learning (Sutton and Barto, 2018). With TD learning, $Q(S_t, A_t)$ is not updated after an episode but after each step. Equation 1.2 shows the update for SARSA, and Equation 1.3 shows the update for Q-learning. Both equations use the observed reward R_{t+1}, from moving from S_t to S_{t+1}, and their version of the estimated reward, $Q(S_{t+1}, A_{t+1})$ for SARSA or $\max_a Q(S_{t+1}, a)$ for Q-learning, to update the quality of the action pair. This process is called bootstrapping because the agent updates $Q(A_t, S_T)$ through another estimation. The one exception is the update at a terminal state; then, the estimated reward will be set to zero.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma\, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{1.2}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{1.3}$$

The difference in the estimated reward between SARSA and Q-learning shows that SARSA is an On-Policy method, and Q-learning is an Off-Policy method (Sutton and Barto, 2018). On-policy methods will try to improve a policy, which also decides which action to pick. SARSA is such a method because it uses the same policy to pick the current action as it did to get the estimated reward. Opposite to this is Off-policy methods; these methods update their policy using a different policy from which it decides its actions. For example, most Q-learning models learn through a version of ϵ-greedy, but their estimated reward, $\max_a Q(S_{t+1}, a)$, is a greedy policy because it picks the quality of the state-action pair if the best action was chosen for the next state.

Another method for updating an agent is Monte-Carlo estimation (Sutton and Barto, 2018). This method uses a collected trajectory τ to calculate the returns and uses the returns to update the agent. This differs from Bootstrapping in that Bootstrapping uses the current reward and the estimated state-action value of the next state.

$$G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{1.4}$$

Equation 1.4 shows how these returns are calculated by the summation of the current reward with the discounted rewards in future states until the final state of trajectory τ. In the equation r_t is the reward found at time step t and γ is the discount factor.

1.1.4 Graph Neural Networks

Besides the multilayer perceptron (MLP), there is a wide range of artificial neural networks specialised in handling different kinds of input data. For example, convolutional neural networks (CNN) and recurrent neural networks (RNN) were introduced to handle images and text input data, respectively. Since then, both have been used on a wide range of input data besides the previous two mentioned. However, both architectures can not handle non-euclidian structured input data, such as graphs. Therefore, Graph Neural Networks (GNN) were introduced.

This section will explain the fundamentals of GNN, which is the method we used to encode our graphs. Besides GNN, there also exist other methods to encode graphs. However, many of these methods are not usable for this research because they function only with unlabelled graphs. For example, methods such as struct2vec (Figueiredo et al., 2017) do not function with labelled graphs and are therefore unusable for other diversity graph problems such as the diversified top-k weighted clique search problem. We need to state that GNNs only function with labelled graphs; however, we can add custom node features to capture the needed structural information of a graph[3]. Moreover, other methods for labelled graphs, such as DeepWalk (Perozzi et al., 2014) and Node2Vec (Grover and Leskovec, 2016), cannot encode essential structural information about the graph. The only other considered option was Structure2Vec (Dai et al., 2016), which can be made to function with unlabelled and labelled graphs, and encodes the structural information of a graph. However, we found that available implementations of Structure2Vec were outdated, and thus we decided to focus on GNNs. We start by explaining the basics of how GNN function and afterwards we state on what kind of problems they are used.

Most GNN architectures function by first collecting the node features of the node itself and its direct neighbour nodes and then passing this through an aggregation function. The aggregation function can be any function but is most commonly a sum or mean pooling function. Next, the GNN passes the aggregated information through a learnable update function. A GNN does this for each node simultaneously. Therefore, the order of the nodes does not matter and means that an GNN is permutation invariant. An essential aspect of an GNN architecture is the number of layers used. With a single layer GNN, the output of a node will only contain the information of the node itself

[3]We will explain how this is done in section 2.5.2 in the literature review.

and its direct neighbours. However, adding more layers will result in the output of a node, including information of nodes further away in the graph. Therefore, an n-layer GNN architecture will include the information of n-hops away from that given node (Sanchez-Lengeling et al., 2021).

We see GNN primarily being used for three kinds of prediction problems on graphs: node-level, edge-level, and graph-level tasks. With node-level tasks, the goal is to identify a node's role within a graph. An example of a node-level task would be to predict the label of a node. Edge-level tasks focus on the interaction between two nodes in a graph by predicting if there is a link or properties of the interaction. Graph-level tasks try to predict the properties of the whole graph (Sanchez-Lengeling et al., 2021). Lastly, there exists a less studied prediction problem, namely subgraph-level prediction problems.

Subgraph-level prediction problems can be categorised as being somewhere between node-level and graph-level prediction problems. Therefore, we see solutions used for those problems also used for subgraph-level tasks. For example, with graph-level tasks, it is common to pool the embedded information of all the nodes after the GNN pass. This method can also be used for subgraph-level tasks by only pooling the information of the nodes in the subgraph (Duvenaud et al., 2015). Another technique based on node-level tasks is to extract the information subgraph's nodes through a GNN and a virtual node linked to all the nodes in the subgraph (Li et al., 2015). Nevertheless, these techniques show a lack of GNN architectures specialised for subgraph-level tasks. Recently there has been more research on such architectures, like SubGNN by Alsentzer et al. (2020). However, we found those architectures to be unproven and challenging to implement at this moment and therefore focused on the two previous mentioned techniques.

Within the research field of GNN architectures, many different kinds of architectures exist, each with its strengths and weaknesses. Cappart et al. (2021) explains this as a three-way trade-off between scalability, expressivity, and generalisation.

1. The scalability of GNN architecture is measured by how well it can handle large graphs with millions of nodes without running into memory problems.

2. A GNN architecture is said to be expressive if it can capture all the essential information of the graph in the output of a node.

3. When a GNN architecture can generalise well, a trained network can achieve similar scores with different structured graphs.

When we try to answer our research question, we must decide how to handle this trade-off when implementing our algorithm.

1.2 Research Question

Our main research question is: **Will a reinforcement learning approach for the diversified top-k clique search problem (DTKC) provide better results than previous**

traditional methods? An RL method will be an improvement if it either gets the highest score or if it gets similar results to previous algorithm, but with less runtime. However, to answer the main research question, this thesis will need to answer the following sub-questions:

- How can we use a GNN architecture to encode the whole graph, and how can we retrieve relevant information about the clique sets afterwards?
- How can we encode the structural information of a graph, such that the RL agent can make a decision about the candidate clique set?
- How well will Deep Clique Comparison Agent (DCCA) generalise and scale between different graphs?
- Could a reinforcement learning method for DTKC not only work for a single value of k but every possible value of k and how does it compare to other algorithms for different values of k?
- Can DCCA be extended to other diversified graph problems, such as the diversified top-k weighted clique search problem (DTKWC) problem?

Chapter 2

Literature Review

The literature review chapter shows an overview of the relevant research field for our proposed method. We first explain models that can generate graphs. We later use those models to generate graphs to train Deep Clique Comparison Agent (DCCA). Afterwards, we explain combinatorial optimisation. In it we state relevant information and related problems to the diversified top-k clique search problem (DTKC). After that, we give an extensive overview of two approaches for DTKC, namely, *EnumKOpt* (Yuan et al., 2015) and *TOPKLS* (Wu et al., 2020). We also explain *TOPKWCLQ* (Wu and Yin, 2021b), an extension of *TOPKLS*, for the diversified top-k weighted clique search problem. Both *TOPKLS* and *TOPKWCLQ* are essential for our research, because we compare DCCA to them.

The following section gives an overview of reinforcement learning algorithms. We omit to discuss deep RL algorithms for continuous action spaces. The reason for this is that our approach has a discrete action space. We also primarily focus on policy gradient algorithms and, in particular, PPO because this is the algorithm we use for our approach. Our next section focuses on graph neural networks and how we can encode graphs as input for our proposed approach. In it, we show what kind of node features we can use, and GIN, the GNN architecture we will use for our approach.

Our last section combines the information of all the previous sections and explains how reinforcement learning is used for combinatorial optimisation problems. We first state how to categorise these methods and explain other relevant concepts. We conclude by detailing some of these proposed methods, how they could be categorised and why they are relevant for our research.

2.1 Graph Generators

There is a subfield within the graph theory research field focused on finding models that can create graphs that hold specific properties. This subsection discusses two graph models, namely the Erdős-Rényi model and the Barabási–Albert model.

The Erdős-Rényi (ER) model (Erdös and Rényi, 1959) generates random graphs, which can either be done through $G(n, m)$ or $G(n, p)$. With $G(n, m)$, the model will

generate a graph with n nodes and a total of m edges. Each possible edge has an equal chance of being generated by the model. The other function, $G(n, p)$, again generates n nodes, but in this model, each edge has a probability of p to be generated. Therefore, if $p = 1$, the model will generate a complete graph, with all possible edges existing in the graph. The reverse happens with $p = 0$ because the model generates a graph with no edges. However, ER models are not realistic to real-world graphs and are unlikely to cluster due to this randomness.

The Barabási–Albert (BA) model (Albert and Barabási, 2002) tries to solve this problem by generating graphs through preferential attachment. The BA model generates graphs through $G(n, m)$, in which n is again the number of nodes in the graph and m is the number of edges from that generated node to other nodes in the graph. The BA model adds these nodes iteratively to the graph. Each added node is then connected to m previous generated nodes, with the probability of that a node picked being higher if it already has many connections. Equation 2.1 calculates this probability for a node, by dividing the degree of that node with the summation of the degrees of all the nodes. Graphs generated by the BA model are more likely to have hubs, which are nodes with a significantly higher degree than the average degree of the graph. These hubs are also seen in many real-world graphs, which indicates that the BA model generates graphs that are more similar to real-world graphs, like social networks.

$$p_v = \frac{d(v, \mathcal{G})}{\sum_{u \in V(\mathcal{G})} d(u, \mathcal{G})} \tag{2.1}$$

There are many extensions to the BA model, such as the extended Barabási–Albert model (Albert and Barabási, 2000) and the Holme and Kim algorithm (Holme and Kim, 2002). This paragraph will discuss one of these, namely the dual Barabási–Albert model (Moshiri, 2018), which we used in the generation of our training and evaluation data sets. The Dual BA model generates graphs by $G(n, m_1, m_2, p)$, again with preferential attachment. However, with Dual BA for each node, either m_1 connections are made with probability p or m_2 with probability $1 - p$. This allows the dual BA model to generate cliques that vary more in size than the original BA model does. We will show this in our analysis of the data sets and graph generator models used for training (section 4.1.1).

2.2 Combinatioral Optimisation

Combinatorial optimisation (CO) problems are problems that have multiple solutions, but only one solution is the most optimal. The solutions for these problems are found by searching through a finite set of objects, and any found solution should satisfy a set of constraints. An objective function then compares the quality of the found solution, and the goal is to either maximise or minimise this objective function.

One of the best known CO problems is the travelling salesman problem (TSP). TSP is not related to diversified top-k clique search problem (DTKC); however, TSP has the most RL algorithms of any CO problems, and thus a basic understanding of TSP is needed to understand its RL algorithms. The goal of TSP is to find the shortest route given a list of cities such that each city on that list is visited at most once. TSP is

not hard to solve with four cities because there are only 25 possible routes; however, if there are ten cities, the number of possible routes grows to 3628800. The reason for this significant growth is that are always $n!$ routes possible with n cities (Laporte, 1992).

TSP shows why it is hard to answer CO problems because finding an optimal solution is done by checking all the possible solutions while the search space grows factorially to the number of cities to visit. CO tries to alleviate this problem, by, for example, decreasing the set of possible solutions or by optimising the search. This research field is too considerable to discuss in its entirety, so this research proposal will only focus on CO problems, problems closely related to DTKC and techniques used to find solutions for these problems.

2.2.1 Local Search

Local Search is a widely used heuristic algorithm that moves through the search space by changing small parts of the solution (Aarts and Lenstra, 1997). How this is done depends on the problem itself, but in most instances, Local Search changes the solution only if it improves some score function. Because of this, Local Search gets regularly stuck at a local optimum. A metaheuristic algorithm can alleviate this problem. Simulated Annealing (Laarhoven and Aarts, 1987), variable neighbourhood search (Mladenović and Hansen, 1997) and evolutionary programming (Ryan, 2003) are examples of metaheuristic algorithms.

2.2.2 Maximal Clique Enumeration

Maximal clique enumeration (MCE) is the enumeration of all the maximal cliques in given graph $\mathcal{G}$. For smaller graphs, MCE is doable in a reasonable amount of time, but MCE does not scale well to the size of graphs for two reasons. The first is the complexity, which grows exponentially because the upper bound of maximal cliques in a graph is $3^{n/3}$ (Moon and Moser, 1965), with n the number of nodes in a graph. This problem can be alleviated by algorithms, like the Bron–Kerbosch algorithm (Bron and Kerbosch, 1973) for dense graphs or the algorithm of Eppstein et al. (2010) for sparse graphs. Nevertheless, these algorithms do not solve the second problem of MCE, which is the problem of saving all the cliques in memory. The space complexity problem is harder to solve, especially for dense graphs. For this reason, solutions for the diversified top-k clique search problem (Yuan et al., 2015; Wu et al., 2020) always have at most k cliques in memory.

Bron–Kerbosch algorithm

As previously mentioned, the Bron–Kerbosch algorithm (Bron and Kerbosch, 1973) enumerates all the maximal cliques in a graph. One of the main benefits of this algorithm is that the algorithm does not have to store any found clique. The Bron–Kerbosch starts with three sets: P, R and X. P contains all the nodes that the algorithm considers for forming a maximal clique. R contains all the nodes that will form the maximal clique. Lastly, X contains all the nodes that the algorithm has already processed. At

the start of the process, P contains all the nodes of the graph, and R and X are empty sets.

Algorithm 1 Bron–Kerbosch algorithm

```
1:  function BronKerbosch(P, R, X, G)
2:      if P = ∅ ∧ X = ∅ then
3:          Report R as a maximal clique
4:      end if
5:      for each v ∈ P do
6:          BronKerbosch(P ∩ N(v, G), R ∪ N(v, G), X ∩ N(v, G))
7:          P ← P \ {v}
8:          X ← X ∪ {v}
9:      end for
10: end function
```

Algorithm 1 shows how the Bron-Kerbosch algorithm is a recursive backtracking algorithm. At the start of the call, it checks if both X and P are empty, and if so, then R is a maximal clique. Otherwise, it checks every node in P to check if it can form a maximal clique by recursively calling itself with as input R, with the node added, and only considering the neighbourhood of that node in the next call. It then removes the node from P and adds it to X. If at a particular call of the algorithm P is empty, but X is not, then it means the clique R is not maximal.

Algorithm 2 Pivot Bron–Kerbosch algorithm

```
1:  function BronKerboschPivot(P, R, X, G)
2:      if P = ∅ ∧ X = ∅ then
3:          Report R as a maximal clique
4:      end if
5:      u ← arg max_{v∈P∪X} |P ∩ N(v, G)|
6:      for each v ∈ P ∪ N(u, G) do
7:          BronKerboschPivot(P ∩ N(v, G), R ∪ N(v, G), X ∩ N(v, G))
8:          P ← P \ {v}
9:          X ← X ∪ {v}
10:     end for
11: end function
```

The main issue of the original Bron-Kerbosch algorithm is that it considers too many non-maximal cliques. For this reason, Tomita et al. (2006) proposed a new version of the algorithm (algorithm 2), in which it does not consider all the nodes in P anymore. They did this by adding a pivot node u, which must come from the set $P \cup X$. Due to pivot node u, algorithm 2 has only to consider nodes in P that are either u or non-neighbours of node u. The pivot node u can be any node in $P \cup X$, but Cazals and Karande (2008) show that the pivot method used in algorithm 2 leads to the best results, and we also see this pivot method in other algorithms (Yuan et al., 2015; Hagberg et al., 2008).

2.2.3 Max k-cover

The goal of the maximum coverage problem, also known as the Max k-cover problem, is to find a subset of k items from a given set, which maximises the coverage. One can formalise this problem as follows: Provided a set $S = \{s_1, s_2, \ldots, s_{m-1}, s_m\}$, find subset $S' \subseteq S$, such that it is $|S'| \leq k$ and maximised for $\left|\bigcup_{s_i \in S'} S_i\right|$. The max k-cover problem has been extended to a wide range of issues, but one important one, for this thesis is the max vertex cover problem (Croce and Paschos, 2012). The objective of the max vertex cover problem is similar to the one of max-k cover, except that now the goal is to find k nodes, which maximise a specific function. The most common of these functions is to maximise the number of edges, thereby finding the k best-connected nodes in a graph.

2.2.4 Maximum Clique Problem

The maximum clique problem (MC) is closely related to DTKC[1] in that the maximum clique is the largest maximal clique in a graph and thus covers the most nodes. The difficulty of this problem comes from the fact that all cliques have to be checked to find the maximum clique. It is important to note that any maximum clique is the maximal independent set in the complementary graph[2].

2.3 Diversified Top-k Clique Search

Previously, we stated the problem statement of the diversified top-k clique search problem (DTKC) and the diversified top-k weighted clique search problem (DTKWC) and discussed other related diversity graph problems[3]. This section will discuss two approaches for DTKC, *EnumKOpt* (Yuan et al., 2015) and *TOPKLS* (Wu et al., 2020), and one for DTKWC, *TOPKWCLQ* (Wu and Yin, 2021b), which is an extension of *TOPKLS*. We start by explaining *EnumKOpt* and afterwards explain both *TOPKLS* and *TOPKWCLQ*, which we will do in one section as both are similar in how they operate.

2.3.1 EnumKOpt

The first ever approach for DTKC is *EnumKOpt* by Yuan et al. (2015), who also defined this problem. This section will explain how they implemented *EnumKOpt*, which they did in multiple versions, that build up to *EnumKOpt*.

Definition 2.3.1. *Private-Node-Set* - Given a set of cliques $\mathcal{D} = \{C_1, C_2, \ldots, C_{k-1}, C_k\}$ in a graph $\mathcal{G}$, and for any $C \in \mathcal{D}$, the private-node-set is the set of nodes, which only occur in clique C and not in any other clique in $\mathcal{D}$.

$$\text{priv}(C, \mathcal{D}) = C \setminus \text{Cov}(\mathcal{D} \setminus \{C\}) \tag{2.2}$$

[1] If $k = 1$, then DTKC is equivalent to the maximum clique problem

[2] A complementary graph $\mathcal{G}'$ is the inverse of a given graph $\mathcal{G}$.

[3] See section 1.1.2

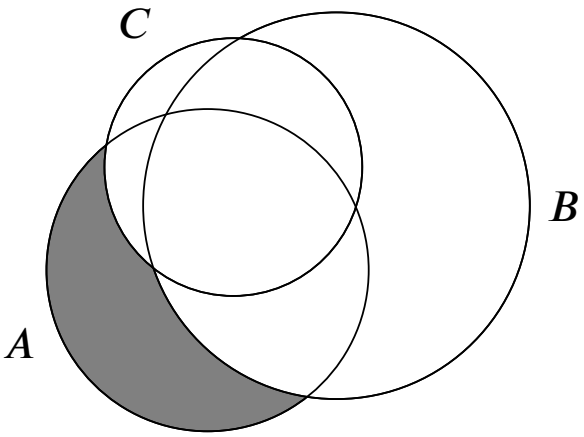


Figure 2.1: This figure shows three cliques: A, B and C. The greyed part in the figure is the private-node-set of clique A. This figure is an example of 2.3.1

Definition 2.3.2. *Min-Cover-Clique* Given a clique set $\mathcal{D} = \{C_1, C_2, \ldots, C_{k-1}, C_k\}$ in a graph $\mathcal{G}$, the Min-Cover-Clique is the clique, which has the lowest amount of private nodes.

$$\mathrm{C}_{\min}(\mathcal{D}) = \arg\min_{C \in \mathcal{D}} \{|\mathrm{priv}(C, \mathcal{D})|\} \tag{2.3}$$

The first version that Yuan et al. (2015) present is *EnumKBasic*. This algorithm modified the MCE algorithm of Eppstein et al. (2010). The original algorithm tries to find all the cliques in a graph, and when it finds a clique, the algorithm adds it to the list of cliques. Yuan et al. (2015) changed this part in *EnumKBasic*, such that there are never more than k cliques stored. When *EnumKBasic* finds a clique, it will first see if the size of the current candidate clique set is smaller than k; if this is the case, it will just add the clique to the candidate clique set. Otherwise, it will compare how many private nodes the found clique has compared to $\mathrm{C}_{\min}(\mathcal{D})$, which needs to be $\alpha \times \frac{|\mathrm{Cov}(\mathcal{D})|}{|\mathcal{D}|}$ better than $\mathrm{C}_{\min}(\mathcal{D})$, with α being a parameter. The function can be seen in algorithm 3.

Yuan et al. (2015) also introduce three other versions of *EnumKBasic*, namely: *EnumK*, *EnumKOpt*, *SeqEnumK* and *IOEnumK*. However, these versions are less important because they only introduce optimisations or are built to function on enormous graphs, with the exception being *EnumKOpt*, which also introduces pruning strategies. This section will give a brief outline of each version, except for *EnumKOpt*, which will be explained more in-depth. The second version, *EnumK*, adds a novel Private-Node-set Preserved Index (PNP-Index). The PNP-Index allow *EnumK* to function far more effective compared to *EnumKBasic* while operating identical. *EnumKOpt* improves *EnumK* by adding three strategies to reduce the number of cliques considered by the algorithm. The first strategy is Global Pruning. With this strategy, each node in the graph is assigned a global priority. The higher a node priority is, the more likely it is that it is a member of a large maximal clique. *EnumKOpt* will find cliques based on the nodes with the highest priority first. *EnumKOpt* will halt if the global pruning score becomes

Algorithm 3 CandMaintainBasic

function CANDMAINTAINBASIC(clique C, clique set $\mathcal{D}$)
  if $|\mathcal{D}| < k$ **then**
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{C\}$
    return $\mathcal{D}$
  end if
  $\mathcal{D}' \leftarrow \left(\mathcal{D} \setminus \left\{\mathrm{C}_{\min}(\mathcal{D})\right\}\right) \cup C$
  if $\left|\mathrm{priv}(C, \mathcal{D}')\right| > \left|\mathrm{priv}(\mathrm{C}_{\min}(\mathcal{D}), \mathcal{D})\right| + \alpha \times \frac{|\mathrm{Cov}(\mathcal{D})|}{|\mathcal{D}|}$ **then**
    return $\mathcal{D}'$
  else
    return $\mathcal{D}$
  end if
end function

lower than $\alpha \times \frac{|\mathrm{Cov}(\mathcal{D})|}{|\mathcal{D}|}$. The second strategy used is Local Pruning. Local pruning lets *EnumKOpt* know if the clique it is currently building still has the potential to improve the candidate clique set. Lastly, Yuan et al. (2015) describe that if the initial candidate clique set of k cliques is of high enough quality, both Global and Local Pruning will perform better. For this reason, they created a method that tries to find k cliques not randomly but in such a way that the coverage of the set is considered. Yuan et al. (2015) built the last two versions, *SeqEnumKOpt* and *IOEnumKOpt*, not to be improvements on *EnumKOpt*, but to function with graphs too large to fit into the main memory.

2.3.2 TOPKLS & TOPKWCLQ

The second method for DTKC is a local search algorithm introduced by Wu et al. (2020) Their paper presents the *TOPKLS* algorithm, which utilises two novel strategies, namely enhanced configuration checking (ECC) and a heuristic that can score the quality of found maximal clique.

The first strategy, ECC, is a modified version of the Configuration Checking algorithm, introduced by Cai et al. (2011), which can prevent cycling the same candidate solution in local search combinatorial optimisation problems (Cai et al., 2015; Li et al., 2016; Wang et al., 2016) and constraint satisfaction problems (Cai and Su, 2013; Abramé et al., 2016). Wu et al. (2020) describe how Configuration Checking did not reduce cycling with DTKC, and thus they had to change the configuration of a node and when the configuration is changed.

Definition 2.3.3. *Configuration ECC* - Given a candidate maximal clique set $\mathcal{D}$ and an undirected graph $\mathcal{G} = (V, E)$, the configuration of a node $v \in V(\mathcal{G})$ is the set $S = \{u | u \in N(v, \mathcal{G}) \setminus \mathrm{Cov}(\mathcal{D})\}$

Definition 2.3.4. *Configuration Change ECC* - Given a candidate maximal clique set $\mathcal{D}$ and an undirected graph $\mathcal{G} = (V, E)$, the configuration of a node $v \in V(\mathcal{G})$ is changed if the set $S = \{u | u \in N(v, \mathcal{G}) \setminus \mathrm{Cov}(\mathcal{D})\}$ has been changed since the last time the node v was removed from $\mathrm{Cov}(\mathcal{D})$.

With these definitions of ECC, *TOPKLS* (Wu et al., 2020) will only consider adding maximal cliques to the candidate clique set, for which all the nodes the configuration has been changed. Therefore, a newfound maximal clique can not contain any nodes for which the configuration has not been changed. Wu et al. (2020) stored the configuration of each node through a Boolean array. If ConfChange $[v] = 1$, the configuration of node v is altered, and ConfChange $[v] = 0$ expresses that the configuration has not been changed. ECC will change the configuration based on the following three rules:

- Rule 1 states: that at the start ConfChange $[v]$ is set to "1" for all the nodes v in the input graph $\mathcal{G}$.
- Rule 2 states: when a maximal clique C is removed from the candidate solution $\mathcal{D}$, then for each $v \in \text{priv}(C, \mathcal{D})$, ConfChange $[v] = 0$ and for each node $u \in (N(v) \setminus \text{cov}(\mathcal{D}))$ is to ConfChange $[u] = 1$, because v has been added to their set configuration.
- Rule 3 states: when a new maximal clique C is added to the candidate solution $\mathcal{D}$, then for each $v \in \text{priv}(C, \mathcal{D})$, ConfChange $[u] = 1$, for each node $u \in (N(v) \setminus \text{cov}(\mathcal{D} \cup \{C\}))$

The *TOPKLS* algorithm finds a clique set through the usage of local search (Wu et al., 2020). This local search runs for a fixed time or until it finds a clique set covering all the nodes in the graph. At each iteration of the algorithm, *TOPKLS* finds an initial set of cliques of size k, which it then starts to improve with local search. For each round of the local search, *TOPKLS* finds a new clique and adds it to the candidate clique set and removes $\text{C}_{\min}(\mathcal{G})$ from the clique set after adding the newfound clique. This order of actions means that $\text{C}_{\min}(\mathcal{G})$ can also be the newfound clique. If the newfound clique set has a better coverage, it will become the new candidate clique set; otherwise, the old candidate clique set stays the candidate clique set in the next iteration of the local search. The local search will stop if the candidate clique set is not improving for several iterations. When this happens, *TOPKLS* will compare this candidate set to the previous one on their coverage and keep the best set. It will then go to the next iteration and repeat the process with a new initial candidate set.

Wu et al. (2020) compared *TOPKLS* to *EnumKOpt* Yuan et al. (2015) on a set of real-world graphs, for $k = 10$, $k = 20$, $k = 30$, $k = 40$ and $k = 50$ and both algorithms have a cutoff time of 600 seconds. The results show that, depending on the graph, *EnumKOpt* and *TOPLKS* either score the same or that *TOPKLS* achieved a higher score. Only on one graph got *EnumKOpt* a better score than *TOPKLS*. However, this comes at a cost of *TOPKLS* having a substantially longer average runtime on each graph than *EnumKOpt*. Wu et al. (2020) also used significantly smaller graphs for their experiments with *TOPKLS* than Yuan et al. (2015) did for their algorithm, which they tested on graphs with 118 million nodes. In contrast, for the experiments with *TOPKLS*, the number of nodes ranged from a few hundred thousand to a few million nodes.

$$\text{C}_{\min}(\mathcal{D}) = \arg\min_{C \in \mathcal{D}} \left\{ \sum_{u \in C} w(u) \right\} \tag{2.4}$$

TOPKWCLQ (Wu and Yin, 2021b) functions similar to *TOPKLS*, with the main difference being the score function. In equation 2.4, we show how *TOPKWCLQ* selects the clique that should be removed from the clique set. With *TOPKLS*, this was the clique with the lowest number of nodes in its private-node-set. However, with *TOPKWCLQ*, this is the clique with the lowest score, which is the summation of the nodes' weights in the clique.

2.4 Reinforcement Learning Algorithms

This section focuses on three kinds of deep reinforcement learning (RL) algorithms: DQN, Policy Gradient, and Neural MCTS. Previously, we discussed in section 1.1.3 essential terminology of RL, which we will use in this section. We mainly focus on Policy Gradient algorithms, and especially PPO (Schulman et al., 2017), because our approach will use PPO as its RL algorithm.

2.4.1 DQN

In section 1.1.3, we briefly discussed Q-Learning and SARSA. Both of these RL methods are Tabular methods, which means that their learned approximated state or state-action values are stored in arrays or tables. These methods work well if the action and state spaces are small enough, such that the agent can easily store them in memory. Still, most action and state spaces are too large for tabular methods. However, researchers have started to combine deep learning methods with RL in recent years, which resulted in deep reinforcement learning. Deep RL utilises deep learning methods to encode the state to an output. The deep learning architecture used depends on the task; for instance, a CNN is used if the input is an image and an RNN for text-based encodings. The main downside of deep RL methods, compared to tabular RL methods, is that it almost always needs more training examples.

One of the most famous deep RL algorithms is deep Q-Learning (DQN) (Mnih et al., 2013). DQN uses a neural network that encodes the current state and outputs the Q-value of each action. This method differs from tabular Q-learning, which stores the current value of each state-action pair. DQN allowed RL to function in environments with an infinite state space. However, without any modification, DQN was too unstable to use. For this reason, two essential modifications were proposed: Experience Replay and Target Networks.

Experience Replay is a memory buffer (Mnih et al., 2013), which stores previous experiences. The DQN agent samples a set of previous experiences from this buffer each time it updates the network's weights, in place of using only the last experience, which the agent adds to the buffer. Each experience is stored in a tuple of $\langle S_t, A_t, S_{t+1}, R_{t+1} \rangle$, with the Experience Replay itself being, in most cases, a First-in-First-out (FIFO) replay buffer and having a set maximum size. The size of the Experience Replay affects the results significantly, with the results dropping if either the buffer is too large or too small (Zhang and Sutton, 2017). DQN benefited greatly from using a memory buffer because it became more stable and became more data-efficient.

$$a_t = \arg\max_a Q(s, a, \theta) \tag{2.5}$$

The other modification to DQN is the usage of two separate weights for the network Q, namely, the standard weights θ and the target weights θ_{target}. A DQN agent uses the standard weights to decide which action to pick through picking an action by equation 2.5, and the agent updates θ after each batch. The agent only uses θ_{target} for calculating the estimated reward for non-terminal states, for which only the found reward is used. This calculation is then used for θ as the loss. The loss calculation can be seen in equation 2.6. The main difference with DQN and tabular Q-learning, is that DQN uses a batch of experiences and thus the expected value of this batch is used as the loss. After a certain number of updates have happened, the agent will do $\theta_{\text{target}} = \theta$. This architecture design made DQN significantly more stable.

$$J(\theta) = \mathbb{E}_{s,a,s',r}\left(r + \gamma Q\left(s', \max_{a'} Q\left(s', a'; \theta_{\text{target}}\right); \theta_{\text{target}}\right) - Q\left(s, a; \theta\right)\right)^2 \tag{2.6}$$

2.4.2 Policy-Gradient Methods

Besides DQN, a value-based method, another kind of Model-Free deep RL method exists, namely policy gradients. The goal of a policy gradients method is to learn a policy $\pi(a|s, \theta)$, with θ being the network weights, to maximise the expected reward. A policy gradients method will thus only output which action to take and not its value. One clear benefit of policy gradient methods over DQN is that they can function in discrete and continuous action spaces, while DQN only functions with discrete action spaces.

One of the oldest policy gradient methods is REINFORCE (Williams, 1992). REINFORCE uses a Monte-Carlo method for training, which means it will play out using $\pi(\cdot|\cdot, \theta)$ and use these experiences to update θ afterwards. Equation 2.7 shows how the gradient is calculated for REINFORCE, which uses the return of a trajectory τ.

$$\nabla_\theta J\left(\theta\right) = \mathbb{E}_\pi\left[G_t(\tau)\nabla_\theta \ln \pi_\theta\left(A_t|S_t\right)\right] \tag{2.7}$$

On its own, REINFORCE proved to be unstable, similar to DQN. A baseline was added to solve this problem. A baseline can be any function, but it should not vary with the chosen actions (Mazyavkina et al., 2021). One common approach for the baseline is to add a second neural network that estimates the value of the current state. However, REINFORCE with baseline still has a high variance because of the Monte-Carlo estimation for training.

Definition 2.4.1. *Baseline* A Baseline b function can be any function that reduces the variance of the policy and consequently should increase the bias. The most common approach for a baseline function is to use a learnable state-value function, $\hat{v}$; however, some algorithms use domain-specific baseline functions.

These value networks are separate networks from the policy network and predict the expected future returns from that state. These value networks use the TD-error δ as the

target for these networks. Equation 2.8 shows how δ is calculated. It is not uncommon for the policy and value networks to share layers, especially if the model is complex.

$$\delta_t = G_t + \gamma \hat{v}\left(s_{t+1}\right) - \hat{v} \tag{2.8}$$

The REINFORCE with baseline combined a policy gradient method with a value-based method. This combination is similar to how another sort of policy gradient method functions, namely, Actor-Critic methods (Konda and Tsitsiklis, 2003). Actor-Critic methods have two networks, an Actor network and a Critic network. The Actor network is the policy and outputs which action to take, while the Critic network outputs a single value, which is an estimation of the current value of that state.

One of the most widely used Actor-Critic methods is A3C (Mnih et al., 2016). A3C introduced two new strategies. The first is that A3C can be trained asynchronously. Therefore, someone can train A3C using only a single global network and have multiple agents collect experience in their environment. The other is using the Advantage for an Actor Critic method. The Advantage tells us how beneficial an action is compared to if a random action is taken at that state. A3C uses this estimated Advantage in the calculation of the loss. The paper of Mnih et al. (2016) also introduces a synchronous version of A3C, named A2C.

The Advantage function

The Advantage function $A(a_t, s_t) = Q(a_t, s_t) - v(a_t, s_t)$ tells how much better an action a_t is at state s_t than all the other actions. Mnih et al. (2016) argues that $\hat{A}(a_t, s_t) = r_t - b_t$ can estimate the Advantage function because r_t estimates $Q(a_t, s_t)$ and b_t estimates $v(s_t)$. When a value network $\hat{v}(s_t)$ is used as baseline, then equation 2.9 can calculate the estimated advantage by using the TD-error from equation 2.8 and a horizon of T. Schulman et al. (2015b) introduced an improved version of this advantage function, named Generalised Advantage Estimation (GAE) (Equation 2.10). They introduced the λ parameter, which helps lower the variance further. If $\lambda = 1$, then GAE is equal to equation 2.9.

$$\hat{A}\left(a_t, s_t\right) = \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t+1}\delta_{T-1} \tag{2.9}$$

$$\hat{A}^{\mathrm{GAE}}\left(a_t, s_t\right) = \delta_t + (\lambda\gamma)\,\delta_{t+1} + \cdots + (\lambda\gamma)^{T-t+1}\,\delta_{T-1} \tag{2.10}$$

Proximal Policy Optimization Algorithms

Policy gradient methods have two significant downsides. The first is that they are sensitive to the learning rate, with a too-small learning rate resulting in almost no progress happening and a too-large learning rate creating much noise and thus instability. The second downside is that policy gradient methods are sample-inefficient. However, a new policy gradient algorithm was introduced to solve these issues, namely, Proximal Policy Optimization Algorithms (PPO) (Schulman et al., 2017). PPO solves the sensitivity to the learning rate by optimising another algorithm, Trust Region Proximal Optimization (TRPO) (Schulman et al., 2015a). TRPO makes use of trust regions to optimise how large of a step it can take without moving too far from its current policy.

TRPO does this by making sure that the Kullback–Leibler divergence D_{KL} (Kullback and Leibler, 1951) between its current policy and a new policy is not too large. However, one major issue of TRPO is that it uses second-order methods, which are highly complex to compute. For this reason, PPO modified this part of TRPO, such that the methods used are first order. PPO tries to be less sample-efficient by training on the data from one or multiple episodes for a certain number of epochs.

Definition 2.4.2. *Probability Ratio* The probability ratio $r_t(\theta)$ is the difference between the current policy and the old policy:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{2.11}$$

PPO does this modification by clipping the policy's loss, such that $r_t(\theta)$ stays close to 1 after each iteration. Therefore, Schulman et al. (2017) proposed the cost function found in equation 2.12.

$$J_t^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}\left[\min\left(r_t(\theta)\,\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{2.12}$$

Equation 2.12 shows how it clips the ratio and then multiplies it by the Advantage explained in section 2.4.2. ϵ is a hyperparameter, which indicates the clipping range, with is in most cases ϵ either 0.1, 0.2 or 0.3. This clipping allows PPO to run multiple epochs on the same data, whereas other Policy Gradient algorithms could not. It also made it so that PPO is more sample efficient and less sensitive to the learning rate. PPO combines this loss with the loss of the value function and an entropy bonus (equation 2.13).

$$J_t^{\text{PPO}}(\theta) = \hat{\mathbb{E}}\left[J_t^{\text{CLIP}}(\theta) - c_1 J_t^{\text{VALUE}}(\theta) + c_2 S\left[\pi_\theta\right]\right] \tag{2.13}$$

In equation 2.13, $J_t^{\text{VALUE}}(\theta)$ is the mean-squared error between the value function and the target, with c_1 being the value coefficient. This coefficient is a hyperparameter and indicates the strength of the value loss in the total loss. Tuning c_1 is essential if the value and policy networks share layers. The addition of an entropy bonus $S\left[\pi_\theta\right]$ helps PPO with exploration, with a higher value c_2 resulting in more exploration.

Another concept that Schulman et al. (2017) introduced with PPO was the idea of early stopping an episode. The main reasoning is that the start of an episode contains enough information for training. PPO also allows multiple agents to collect data concurrently, after which PPO updates the network's weights.

Lastly, it is expected that during training, PPO selects actions stochastic, which means that they are picked based on a probability distribution that it learns. This is because it helps with exploration during training and is how PPO handles the exploration-exploitation trade-off, which we discussed in section 1.1.3. In most research, PPO selects actions deterministic, meaning it will always select the best action during the model evaluation. It is also possible for PPO to select actions stochastic during the evaluation. However, we will set PPO to select actions deterministic when we do the experiments for our algorithm.

2.4.3 Neural MCTS

All the previously explained deep RL algorithms are model-free algorithms, which means they do not know the workings of the transition function. However, some recent breakthroughs showed how model-based deep RL could function and how it can even learn the rules of the environment. AlphaGo (Silver et al., 2016) popularised the model-based approach for deep RL. Silver et al. (2016) implemented AlphaGo through a combination of policy and value networks and MCTS. However, it still had some domain knowledge of the game Go programmed into it, needed human data and needed to know the rules of Go. Later research improved upon AlphaGo, with AlphaGo Zero (Silver et al., 2017b) removing the need for human data and domain knowledge and AlphaZero (Silver et al., 2017a) also being able to play chess and shogi. The latest version, MuZero (Schrittwieser et al., 2019), can even learn the rules of a game and learn how to play Atari games. Recent applications of this algorithm can accurately predict human protein structures, which is a challenging task (Tunyasuvunakool et al., 2021; Baek et al., 2021). However, these algorithms are currently unfeasible in most cases because of the required hardware, with MuZero being trained on 40 TPU's (Schrittwieser et al., 2019).

2.5 Graph Neural Networks

In this section, we start by explaining the essential information about Graph Isomorphic Networks (GIN). As previously mentioned, in our background explanation of Graph Neural Networks (GNN)[4], we need to decide how we want to handle the three-way trade-off between scalability, expressivity, and generalisation. Due to the novelty of our approach, we decided to focus on the expressivity of an GNN architecture. Therefore, we decided to use GIN instead of other architectures, such as Graph Convolutional Networks (Kipf and Welling, 2017) and Graph Attention Networks (Veličković et al., 2017), which are known to scale and generalise better.

Next, we will explain how to select custom node features such that an GNN architecture can encode more information about the graph in its outputs.

2.5.1 Graph Isomorphism Network (GIN)

The focus on the expressiveness of a GNN architecture made us decide to use a Graph Isomorphic Networks (GIN), such that the network can encode the needed structural information. Xu et al. (2018a) developed GIN with the hypothesis that GNN could differentiate between graph structures similar to the Weisfeiler-Lehman (WL) test (Shervashidze et al., 2011).

Xu et al. (2018a) wanted to develop a GNN architecture, GIN, with the same strength as the WL-test. Therefore, they asserted that GIN should output the same outputs for two graphs if the WL-test state that those graphs are isomorphic. They also asserted that previous GNN architectures, such as GraphSage (Hamilton et al., 2017) and GCN (Kipf and Welling, 2017), were not potent enough for this task.

[4]See section 1.1.4

The primary reason for this is the shallow update function of previous GNN architectures. The update function consisted, in most cases, of a single layer update function, which is not powerful enough to simulate a WL-test. Therefore, they added an MLP update function within GIN.

$$h_v^{(l)} = \text{MLP}\left((1+\epsilon)\cdot h_v^{(l-1)} + \sum_{u\in N(v,\mathcal{G})} h_u^{(l-1)}\right) \tag{2.14}$$

Equation 2.14 shows how GIN aggregates the information of the node ($h_v^{(l-1)}$) and its neighbours ($\sum_{u\in N(v,\mathcal{G})} h_u^{(l-1)}$) from the previous layer ($l-1$). GIN does this aggregation by summation because, according to Xu et al. (2018a), doing a mean or max aggregation would result in too much information lost. ϵ in equation ref is used to strengthen or lessen the influence of the own node in the aggregation. Xu et al. state in their paper that ϵ is a learnable value. However, they found that the best results in most test cases were with $\epsilon = 0$ and it not being learnable.

Xu et al. (2018a) also proposed how to design a "readout" function for GIN networks for graph-level tasks, which by association should also work for subgraph-level tasks. Their proposition states that both a mean and max aggregation function will result in losing essential structural information. They state that only a sum aggregation of the nodes encodings will capture enough structural information.

2.5.2 Node Attributes

As previously mentioned, GNN architectures are primarily used for node-level, graph-level or edge-level tasks, in which either the node or edge itself has features. However, most graph CO problems, like DTKC, have no features or only weights, like DTKWC, as features. Therefore, adding features to either the node or the edges is necessary.

Some research is done on which features can help GNN capture the necessary information. Cui et al. (2021) describe some node features that can be added to nodes in a graph and separates them into two distinct groups: positional features and structural features. Positional features help a GNN capture information about the position of nodes in a graph, such that it can learn how close two nodes are together in the graph. Examples of custom positional node features are the output of DeepWalk (Perozzi et al., 2014) and random node features.

Structural features help GNN learn the role of a node in a graph. Therefore, if two nodes are far from each other in a graph but have similar roles, for example, being hub-nodes, then with structural node features, GNNs should be able to learn to place them in the same class. Cui et al. (2021) states that one-hot encoding of the degree and the PageRank (Page et al., 1999) of the node could act as structural node features. Another structural feature is a shared vector of all 1, which is then used by all the nodes (Errica et al., 2019). This shared vector can only help identify structural information if the aggregation is a summation. With a single GNN layer and a shared vector, the GNN can identify how many neighbours each node has because the summation is the number of neighbours. With each subsequent GNN layer, this information is passed on to its neighbours, and the node receives more information about its neighbours.

2.6 Reinforcement Learning in Combinatioral Optimisation

Recently, an uprise in research of utilising reinforcement learning (RL) for combinatorial optimisation (CO) problems has happened. Before it, researchers used mainly supervised methods for CO problems with specialised networks like Hopfield Networks (Mańdziuk, 1996; Liang, 1996) and, more recently, Pointer Networks (Vinyals et al., 2015). However, these algorithms could not scale and did work well with unseen data. The paper of Bello et al. (2016) introduced a framework for utilising RL for CO and coined the term neural combinatorial optimisation (NCO) for machine learning for NCO[5]. This section will first explain some definitions and training strategies used in NCO-RL. Afterwards, we describe some current state-of-the-art NCO-RL approaches.

The survey paper of Mazyavkina et al. (2021) shows examples of how RL can be integrated for CO problems. Mazyavkina et al. (2021) does this by dividing approaches into different categories. The first category divides RL methods that finds solutions on its own, which they named *Principal* learners from those that improve the workings of another solver and thus is *Jointly* trained. Secondly, they distinguish between RL methods, which find their solution through a *Constructive* heuristic or improve an existing solution through an *Improve* heuristic. With the *Constructive* heuristic, a RL method will build a solution until it is a valid solution. If a method uses an *Improve* heuristic, it starts with a valid solution and improves the existing solution.

Another essential aspect of NCO is how each method is trained (Bello et al., 2016). The most common approach, especially for deep RL, is to pretrain an RL algorithm on different instances of the CO problem. The instances can either be real-world data or, more common, randomly generated data, like through the Erdős-Rényi model (Erdös and Rényi, 1959) and the Barabási–Albert model (Albert and Barabási, 2002). The reason why an RL agent can learn on randomly generated data is that the goal function of a CO problem can easily be translated to the reward function of an RL agent. An RL agent can also train directly on the CO problem that needs to be solved. We see this exclusively done with tabular RL algorithm because they are significantly faster to train. This retraining of tabular RL algorithms allows those algorithms to generalise better between different instances of a CO problem. However, this also increases the runtime significantly because those algorithms need to retrain for each instance.

Cappart et al. (2021) describe two benefits of NCO. The first, the describe, is the *encode-process-decode* paradigm (Hamrick et al., 2018), and explains how it can be utilised in NCO. They state how one network can encode the inputs of a CO problem to a latent encoding $\mathcal{Z}$. This latent encoding can then be reused, which can alleviate the scalability problem of NCO. Another potential advantage of this paradigm is the potential for multi-task learning. For example, in the context of DTKC, both a *Clique Finding Agent* and *Clique Comparison Agent* use the same latent encoding.

Secondly, Cappart et al. (2021) state an interesting promise of NCO, namely, its potential to work with natural inputs and data that has non-linear relations, something classical CO algorithms struggle with. A common problem that limits CO algorithms

[5]NCO-RL will be used in this proposal for the research field of reinforcement learning for combinatorial optimisation

from functioning on real-world problems is their need to function with abstractified input data. This abstractification is mainly done manually and therefore can miss latent information about, for example, the weight between two nodes. NCO should overcome this; however, research lacks on how to use natural inputs and non-linear relations with NCO (Cappart et al., 2021).

2.6.1 Recent developments

Although travelling salesman problem (TSP) is not related to DTKC, it is the most widely studied CO problem in NCO-RL research. Therefore, it has the most NCO-RL algorithms, which can act as examples of the different NCO-RL categories of Mazyavkina et al. (2021), which were introduced in the introduction of this section. Bello et al. (2016) was the first to create an RL agent for TSP, which used the then-recently proposed Pointer Network architecture to encode the input space. The agent started from an arbitrary starting node, from which it picked the next state until each node in the instance was visited. The proposed algorithm of Bello et al. (2016) is an example of a *Principal* learner with a *Constructive* heuristic. After the algorithm of Bello et al. (2016), proposed methods used similar techniques, with the only significant improvement being the inclusion of Attention (Deudon et al., 2018; Kool et al., 2019). Chen and Tian (2019) proposed an algorithm that improves an existing TSP solution until convergence. Their algorithm also worked for the Job-Shop Scheduling Problem and other routing problems and outperformed existing non-NCO solutions. Cappart et al. (2020) gives another example of *Jointly* trained RL algorithm. Their paper proposes an RL algorithm that improves Constraint Programming, which can solve a wide range of CO problems. Cappart et al. (2020) also noticed how Constraint Programming is linked to Dynamic Programming. The previously shown examples of NCO-RL are all pretrained deep RL algorithm. However, a later paper (Zheng et al., 2020) states how deep RL struggles to scale to larger problem instances of TSP. In their paper, Zheng et al. (2020) proposes a tabular agent, which improves the Lin-Kernighan-Helsgaun algorithm. Their algorithm outperformed deep RL algorithms by a significant margin and had no problems with scaling.

Currently, no RL algorithm exists for DTKC; however, there are many approaches for finding solutions for the maximum clique problem (MC) and the maximum independent set problem (MIS)[6]. Abe et al. (2019) proposed one of the first approaches for the maximum clique problem, which could also be trained for other NP-Hard graph problems. Their proposed algorithms uses Neural MCTS with an GIN architecture to encode the graphs, which is a *principal* learner that *improves* existing solutions. They tested this algorithm on some real-world graphs and found that it could find comparable results to previous approaches and even found maximum cliques for some graphs that surpass previous best-known solutions. However, they only tested it on small graphs, with at most 5000 nodes. This is likely due to GIN not being scalable, but they state that another GNN architecture could replace GIN in future approaches. Another algorithm is the one we previously described by Cappart et al. (2020), which uses RL to enhance the functioning of Decision Diagrams and encodes the graph with Struc-

[6]In section 2.2.4, we explained MC and MIS, and how they are related to DTKC

ture2Vec (S2V) (Dai et al., 2016). Therefore, this algorithm is a *joint* learner, because it improves the workings of another algorithm, which means it is not relevant for our approach, because we cannot utilise Decision Diagrams for DTKC. Lastly, the algorithm of Ahn et al. (2020) tried to alleviate this scalability problem, that limited Abe et al. (2019). Ahn et al. (2020) described how they designed an RL agent with PPO and a GCN as encoder, such that the algorithm could learn to defer. This deferring meant that the agent decided at each transition to either include, exclude or defer a node in its solution, with deferring meaning that the algorithm decides at a later stage of the node will be included or excluded. Ahn et al. (2020) tested their approach on graphs with a maximum size of two million nodes. The results show that the algorithm of Ahn et al. (2020) outperformed not only other NCO-RL algorithms but also classical algorithms.

Although there is no paper on utilising RL on a diversified graph problem, there is a paper that utilises RL for a diversified top-k recommender system (Zou et al., 2019). The algorithm used neural MCTS, based on AlphaGo (Silver et al., 2016), for the RL agent. However, the most critical aspect of this paper for DTKC is how actions are rewarded. Zou et al. (2019) designed the reward function in such a way that it rewards diversification.

The examples given in the previous paragraphs show the potential of RL on CO problems. Nevertheless, they are still focused on the theoretical sides of the problem. However, recently a practical breakthrough application was proposed (Mirhoseini et al., 2021). This paper by Mirhoseini et al. (2021) proposes an NCO-RL agent that can design TPU chips, which is a CO problem. The agent created chip designs significantly faster than humans do[7]. These designs were also similar in quality.

[7]The agent needed only about six hours for a design, while a human team would take at least weeks.

Chapter 3

Methodology

We showed in the literature review that DTKC consists of two steps: finding a maximal clique and then deciding if the found clique should replace a clique in the current candidate clique set. We decided to focus on the second step and implement an RL algorithm that learns how to compose the best clique set, with the clique finding being done by another algorithm. Therefore, we propose the Deep Clique Comparison Agent (DCCA), which can learn to find the ideal clique set depending on the diversity graph problem and not only for DTKC.

We decided to use reinforcement learning because of the infinite number of training graphs we can generate through the dual BA model. Supervised learning needs to have labelled data, which does not exist for DTKC or DTKWC. Approaches, such as *TOPKLS* and *EnumKOpt* (Wu et al., 2020; Yuan et al., 2015), could generate this data. However, this data will likely not be the exact solution because both algorithms find their solution through approximation. Thus, a supervised model trained on this data will likely make the same mistakes as *TOPKLS* and *EnumKOpt*. This data could be combined with Imitation Learning, but this again will limit an algorithm to only problems that already have an existing algorithm (Cappart et al., 2021).

The literature section stated that almost every CO problem has a discrete observation and action space. In theory, this would mean that a Tabular RL algorithm could learn a CO problem. Nonetheless, the number of possible states is too expansive for any Tabular RL algorithm to learn. Hence, we decided to leverage deep learning methods, especially GNNs, to overcome this problem.

This chapter will discuss our proposed algorithm and design choices. We start by how the diversified top-k clique search problem and other diversity graphs problems can be formulated as a Markov decision process (MDP).

3.1 MDP

We can formulate the diversified top-k clique search problem (DTKC) or any related problem, such as the weighted variant, as a Markov Decision Process. The following Markov decision process (MDP) shows how this formulation:

- **State:** Each state s_t consists of the current candidate clique set $\mathcal{D}_t$ with the newfound clique set C_t. Therefore, a state is $s_t = \mathcal{D}_t \cup \{C_t\}$
- **Action:** The action space is discrete and always consists of $k + 1$ possible actions. The action a_t signifies which clique will be removed and replaced by the newfound clique, except when $a_t = k + 1$, which is used to signify that the newfound clique will not replace a clique.
- **Transition:** The transition function removes the selected clique from the state and adds a new clique found. Therefore, the function is as follows: $T\left(s_{t+1}|s_t, a_t\right) = s_t \setminus \left\{\left(\mathcal{D}_t \cup \{C_t\}\right)_{a_t}\right\} \cup \{C_{t+1}\}$
- **Reward Function:** The reward function is the difference in score between the next state and the current state. The score function differs between problems, but for DTKC, it is the coverage of the clique set. We will explain the reward function in-depth in subsection 3.1.1.

To find all the cliques in a graph, we use the Pivot Bron–Kerbosch algorithm[1] (Cazals and Karande, 2008). We decided to use this algorithm for two reasons. The first reason is that this algorithm is deterministic; therefore, the ordering of the cliques will always be in the same order for a given graph, and thus, the MDP can model the state transition. For instance, the clique finding algorithms used by *TOPKLS* (Wu et al., 2020) and *TOPKWCLQ* (Wu and Yin, 2021b) are stochastic. The second reason is that it can be used for both DTKC and DTKWC. This is the reason is why we cannot use the clique finding algorithm of *EnumKOpt* (Yuan et al., 2015) because it only functions for DTKC.

3.1.1 Reward Function

The design of the reward function took more time than initially planned. There are two reasons for this, which we will discuss in this section and how we overcame it such that we can defend the final reward function.

At first sight, the reward function should be the score function of DTKC or any other diversified graph problem. However, the first issue with this reward function is that the action taken does not influence the score enough. This problem correlates to the parameter k, with example 3.1 showing this correlation.

Example 3.1. If our reward function is the coverage of the current clique set and $k = 50$, then a single clique will only have a 2% influence on the score compared to the other cliques. With $k = 10$, this influence would increase to 10%.

Therefore, the reward function should be the difference in score between time step $t + 1$ and t and thus is $r_t = \text{score}(s_{t+1}) - \text{score}(s_t)$. The difference in score gives more information about the individual action, and through an high enough γ, it should also help decide on future actions. The main issue with using this reward function is that the range of possible rewards is enormous; depending on the graph, the range of rewards

[1] See algorithm 2 in section 2.2.2

can easily be between -100 and 100. Nevertheless, there are two possible solutions to solve this problem.

The first is to divide the reward by the maximum clique for DTKC or the maximum weighted clique for DTKWC. There are algorithms that find these cliques either precisely or through approximation (Boppana and Halldórsson, 1992; Warren and Hicks, 2006).However, due to both problems being NP-Hard, finding these cliques can be computationally heavy depending on the graph and therefore slow down training significantly when graphs are generated during training. Another negative of this solution is that it only can be used for DTKC and DTKWC. Therefore, we decided to focus on another solution.

The second solution is to scale the reward by a scalar value ρ such that the reward range stays closer to 0. Cappart et al. (2018) proposed this solution for their deep RL algorithm for the maximum cut-problem and the maximum independent set. They argued that it improved training because gradient descent struggles with sparse and large rewards. We also decided to implement this scaling for our algorithm because it allows the agent to learn other problems than DTKC and DTKWC. The final reward function is thus equation 3.1, with ρ being the scalar value:

$$r_t = \rho \left(\text{score}(s_{t+1}) - \text{score}(s_t) \right). \tag{3.1}$$

The last two equations show the specific reward function for each problem we will train DCCA for. The reward function for DTKC (equation 3.2) is the difference between the size of the new coverage and the old coverage. For DTKWC (equation 3.3), this is the difference between the summation of the weights between the old and new coverage.

$$r_t^{\text{DTKC}} = \rho \left(\left| \text{Cov}\left(\mathcal{D}_{t+1}\right) \right| - \left| \text{Cov}\left(\mathcal{D}_t\right) \right| \right) \tag{3.2}$$

$$r_t^{\text{DTKWC}} = \rho \left(\left(\sum_{v \in \text{Cov}(\mathcal{D}_{t+1})} w(v) \right) - \left(\sum_{u \in \text{Cov}(\mathcal{D}_t)} w(u) \right) \right) \tag{3.3}$$

3.2 Network Architecture

Our algorithm DCCA, encodes the graph and the cliques by using two graph neural network architectures, one for encoding all the nodes in the graph and the second architecture which acts as the actor and critic network for the PPO algorithm. This section will first discuss each architecture separately and conclude by explaining how the two interact. Figure 3.1 shows how the networks are connected.

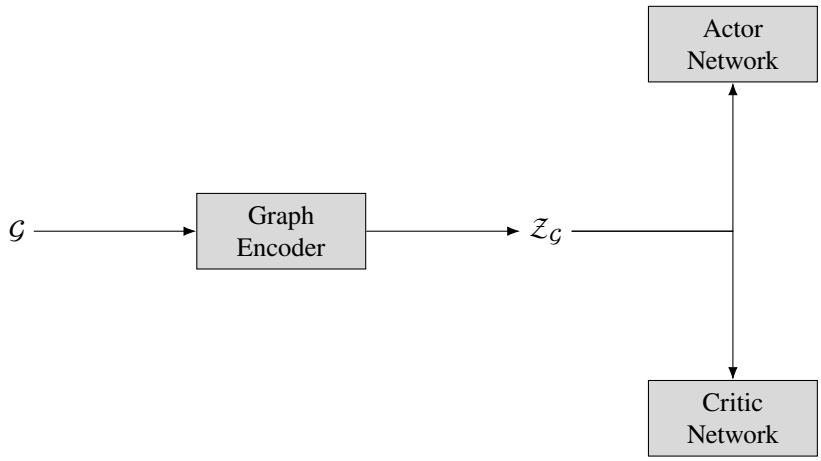


Figure 3.1: This figure show our network design. We will explain how the Graph Encoder functions in section 3.2.1. This networks gets as input a graph $\mathcal{G}$ and outputs the latent node encodings $\mathcal{Z}_{\mathcal{G}}$. These encodings are then used at each step in the episode by the actor and critic network, which we explain in section 3.2.2

3.2.1 Graph Encoder

The task of the first network is to encode the whole graph such that the structural information of the graph is encoded into latent vectors for the nodes. Therefore, we needed to find a GNN architecture that would capture this information and select input features, which help the GNN architecture capture this information. We decided to use a Graph Isomorphic Networks (GIN) (Xu et al., 2018a) as our GNN architecture, based on the usage by Abe et al. (2019). They demonstrated that an RL agent could learn to find the maximum clique in a given graph using a GIN.

In their paper, Abe et al. (2019) used five layers of GIN with a hidden dimension of 32 and each MLP. The GIN layers also consisted of five layers, in which the input and output dimensions were thus 32 and the hidden dimension 16. As input, they used a vector of ones, which helped capture the structural information of the graph. Other research also shows that this method can capture the relevant structural information (Cui et al., 2021).

For DCCA, we decided to use a similar network setup as Abe et al. (2019). We decided to test different numbers of GIN layers and hidden dimension sizes for both the GIN layer and the MLP within the GIN layer. We will state the final setup in our hyperparameter section (table 4.8).

The graph encoder will encode the latent representation of the nodes $z \in \mathcal{Z}_{\mathcal{G}}$ of a given graph $\mathcal{G}$. The actor and critic network use $\mathcal{Z}_{\mathcal{G}}$ as their input at each step. We based this setup on the *encode-process-decode* paradigm (Cappart et al., 2021), which states that multiple computations can be done on the same latent space.

The main downside of using GIN is that the architecture is computational heavy compared to other architecture such as Graph Convolutional Networks (Kipf and Welling,

2017) and Graph Attention Networks (Veličković et al., 2017). However, the GIN encoder network is only run once for each graph, and thus this computational heaviness is insignificant during the evaluation, but it does increase the training time.

3.2.2 Actor-Critic Network

Both the actor and critic network use a GIN architecture. Therefore, we decided to use the virtual node method for our subgraph-level task. The upcoming paragraphs will explain both networks and the inputs for them.

Actor Network

The actor network has as input each clique and thus uses $k + 1$ virtual nodes, one for each clique in the current candidate clique set D_t and the newfound clique C_t. It is essential to state that the input of each clique node is independent of one another. This independence means two virtual nodes can share the same nodes as input, but they do not communicate. Figure 3.2 shows an example of this procedure.

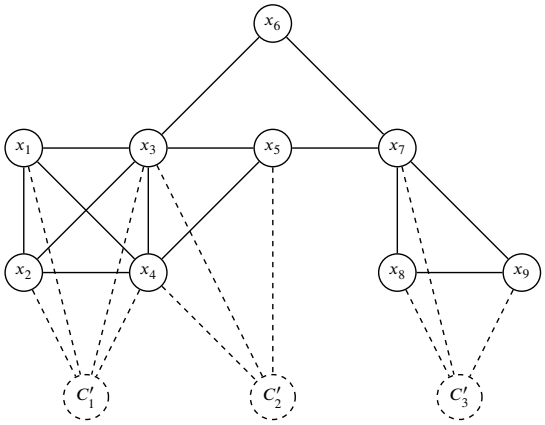


Figure 3.2: This figure shows the actor input for three different cliques, with $C_1 = \{x_1, x_2, x_3, x_4\}$, $C_2 = \{x_3, x_4, x_5\}$ and $C_3 = \{x_7, x_8, x_9\}$. C_1 and C_2 share the nodes x_3 and x_4, and therefore both have their latent encoding $z_i \in \mathcal{Z}_\mathcal{G}$ as input.

Equation 3.4 shows the input for the actor network. The 0 in the calculation is normally the node itself, but because we use virtual nodes, this will be 0. The actor network collects the latent node encodings $z_u \in \mathcal{Z}_\mathcal{G}$ from a clique $u \in C_i$. It does this for all the cliques in the current candidate clique set $C \in D_t$ and the newfound clique C_t. Therefore, the final output of the actor network is $X \in \mathbb{R}^{(k+1)}$.

$$C'_i = \text{MLP}\left((1+\epsilon)\cdot 0 + \sum_{u \in C_i} z_u\right) \tag{3.4}$$

Critic Network

The critic network uses a single virtual node with $\text{Cov}(\mathcal{D}_t \cup \{C_t\})$ as input. This virtual node outputs the value of that state. This method is based on the algorithm of Zhang et al. (2020), which used a similar setup for their critic network. Figure 3.3 shows an example of this process.

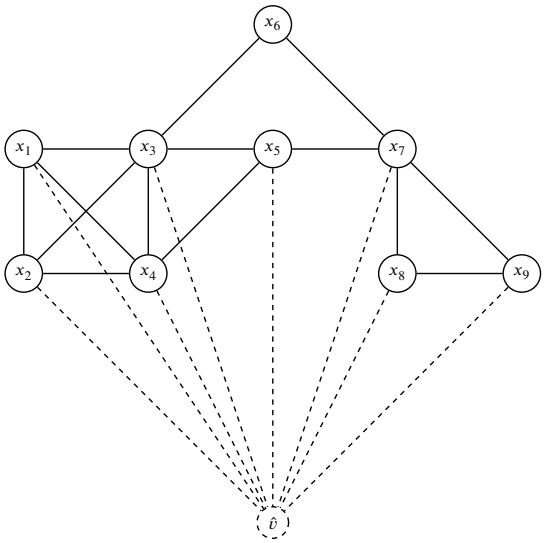


Figure 3.3: The figure shows how $\hat{v}(s_t)$ is calculated by collating the latent encodings of the cliques in figure 3.2, with $s_t = \{C_1, C_2, C_3\}$

Equation 3.5 shows the input for the critic network. The input of this is the coverage of the current candidate clique set $\mathcal{D}_t$ and the newfound clique C_t. The critic network collects all the latent node encodings $z_u \in \mathcal{Z}_G$ from all the nodes found in the coverage of the current state $u \in \text{Cov}(s_t)$, which is $\text{Cov}(s_t) = \text{Cov}(\mathcal{D}_t \cup \{C_t\})$. The final output is then a single value $\hat{v}(s_t) \in \mathbb{R}$

$$\hat{v}(s_t) = \text{MLP}\left((1+\epsilon)\cdot 0 + \sum_{u \in \text{Cov}(\mathcal{D}_t \cup \{C_t\})} z_u\right) \tag{3.5}$$

3.3 Deep Clique Comparison Agent (DCCA)

This section will discuss how implemented Deep Clique Comparison Agent (DCCA), for which we used PPO. We start by showing how DCCA executes a single episode. Afterwards, we explain the training procedure of DCCA. Lastly, we state our argumentation of why we chose to use PPO over other reinforcement learning algorithms.

Algorithm 4 The clique composing operation of DCCA

function COMPOSECLIQUESET(Graph $\mathcal{G}$, actor weights θ_a, graph encoder weights $\theta_{\mathcal{G}}$)
  $\mathcal{D} \leftarrow \emptyset$ ▷ Initialise an empty candidate clique set
  $\mathcal{Z}_{\mathcal{G}} \leftarrow \theta_{\mathcal{G}}(\mathcal{G})$ ▷ Gather the node latent features for the whole graph
  for $C \in$ PivotBronKerbosch($\mathcal{G}$) **do**
    if $|\mathcal{D}| < k$ **then** ▷ The first k cliques are added
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{C\}$
    else
      $\mathcal{D}^* \leftarrow \mathcal{D} \cup \{C\}$
      $a \leftarrow \pi_{\theta_a}(\mathcal{D}^*, \mathcal{Z}_{\mathcal{G}})$ ▷ The actor decides which clique should be removed
      $\mathcal{D} \leftarrow \mathcal{D}^* \setminus \{\mathcal{D}^*_a\}$
    end if
  end for
  return $\mathcal{D}$
end function

In algorithm 4, we show how DCCA composes a clique set from a graph. At the start of the run, the algorithm initialises an empty candidate clique set. It uses the graph encoder network to generate the latent node encodings $\mathcal{Z}_{\mathcal{G}}$ for the whole graph, which we reuse as input for the actor-network at each iteration. Cappart et al. (2021) argues how the *encode-process-decode* paradigm can be used for algorithmic reusing, which we do, or multi-task learning, for which we make recommendations in our discussion. This reusing of the latent encodings allows us to speed the execution time of DCCA significantly because calculating the node latent encodings of the whole graph is the computationally heaviest task and should improve the scalability of DCCA.

We use the Pivot Bron-Kerbosch algorithm (Cazals and Karande, 2008) to find all the maximal cliques in the graph. The algorithm will always add the first k cliques to the candidate clique set $\mathcal{D}$. When $|\mathcal{D}| = k$, a decision clique set $\mathcal{D}^*$ is created by adding the newfound clique to $\mathcal{D}$. The actor-network then uses $\mathcal{D}^*$, and the latent node features $z_n \in \mathcal{Z}_{\mathcal{G}}$ as input to decide which clique should be removed from $\mathcal{D}^*$. This clique is then removed, and $\mathcal{D}$ becomes $\mathcal{D}^*$ with the removed clique. This process repeats until Deep Clique Comparison Agent (DCCA) checked every clique in graph $\mathcal{G}$, at which point the current candidate clique set will be returned as the final clique set.

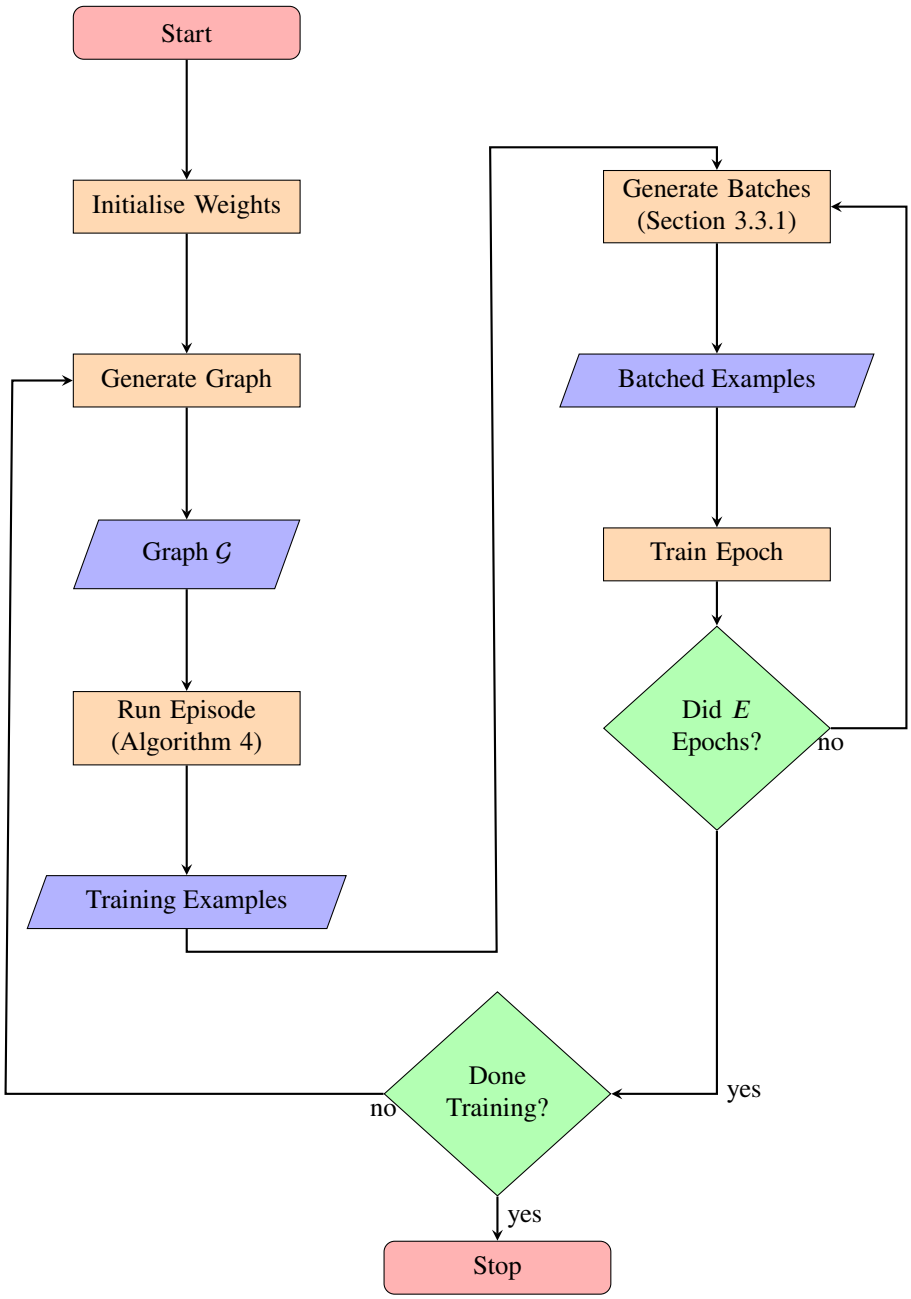


Figure 3.4: The flowchart of the training procedure of DCCA.

Figure 3.4 shows the training procedure of DCCA. We start by initialising the weights of all the networks, which are the graph encoder, actor and critic network. After that, the training loop starts. At the beginning of each loop, we will generate a graph. This graph is then used to create training examples by running an episode using algorithm 4. We generate batches from those training examples with a custom batching algorithm, which we will explain in section 3.3.1. Those batches are shuffled at the start of each epoch. DCCA will then train E epochs on those batches. After that, it will either start a new episode with a newly generated graph or stop training, if it trained for a certain threshold of steps.

Besides PPO and other policy gradient algorithms, the literature review chapter also discussed two different deep RL algorithms: DQN and Neural MCTS. Both DQN and Neural MCTS would have been good choices; however, they are significantly slower to train or take many more resources.

Compared to PPO, DQN is, in most cases, slower to train while being more sample efficient and therefore is the better choice if gathering data is computationally heavy. For DTKC and DTKWC, this is not the case because, for both problems, we can gather unlimited data without it being computationally heavy to do. DQN is comparably even slower for our approach than PPO because PPO can gather data from each episode by reusing the output of the graph encoder $\mathcal{Z}_{\mathcal{G}}$. With DQN, this would not be possible because DQN does gradient ascent after each action taken, thus changing the weights for the graph encoder.

We decided not to use Neural MCTS because it is slower to train, similar to DQN, and needs significantly more resources. Moreover, due to Neural MCTS being a recent development, there are few implementations, and thus it would be considerably harder to implement.

3.3.1 Batching Algorithm

Most batching methods for training a GNN architecture can be divided into two groups, one for graph-level tasks and one for node-level tasks. With batching for graph-level tasks, the batching algorithm will combine multiple graphs into a single graph for a single forward pass. Node-level tasks combine multiple node-level tasks for a single graph into one pass. For example, it will try to predict the node labels for all the nodes in a single forward pass. However, DCCA is a subgraph-level task, for which no batching algorithm exists. Therefore, we decided to design a new batching algorithm to improve the training time significantly.

As previously mentioned, we collect $k+1$ clique outputs for the actor network and a single coverage output for the value network through a GNN architecture from a latent encoded graph. This new batching algorithm aims to have a single pass of the graph encoder network for the graph and then perform B steps on this latent encoded graph, in which B is the batch size. Therefore, the batching algorithm collects B steps, which were performed on the same graph, such that $B(k+1)$ cliques inputs and B coverage inputs are collected. The network's output is thus two vectors, $\mathbf{X} \in \mathbb{R}^{B(k+1)\times 1}$, the output of the actor network and $\mathbf{V} \in \mathbb{R}^{B\times 1}$, the output of the value network. X does not have the correct output dimensions because we want to have the cliques of the same step in a single row, such that the $B(k+1)\times 1$ vector is transformed into a $B \times k+1$ matrix.

The algorithm, therefore, creates a batch index vector $I \in \mathbb{Z}_{+}^{B(k+1)\times 1}$ that collects the cliques' outputs of a single step in a row such that we get a $B \times (k+1)$ matrix.

3.3.2 Software and Hardware

We implemented DCCA in Python with the use of PyTorch and Pytorch Geometric (Paszke et al., 2019; Fey and Lenssen, 2019). We used NetworkX to find the cliques in a graph (Hagberg et al., 2008). We did the experiments with a Ryzen 5 2600 Six-Core Processor 3.4 GHz CPU, 16GB RAM and an Nvidia GeForce RTX 2060 GPU. We based our algorithm on the code from Kostrikov (2018)[2].

3.4 Closing Remarks

This chapter discussed our proposed algorithm named DCCA. We showed how DTKC and other diversity graph problems could be formulated as an MDP and how to approach the reward function. After that, we explained the network design and how DCCA only has to calculate the latent encodings of a graph once, such that the actor and critic network can reuse them to compose the clique set. Later in the thesis, we will show that reusing is possible through experiments. If possible, this reusing would be not only beneficial for DCCA but also for other NCO-RL algorithms because it could help lessen the scalability problem of NCO-RL.

We also discussed how the actor and critic networks use virtual nodes and GIN. It is essential to state that because the virtual nodes are 0, the GIN operation is theoretically identical to sum pooling the node encodings and passing them through an MLP. However, this method allows future research to change the GIN architecture to other GNN architectures.

The last section states how implemented PPO for DCCA. We explained how each step DCCA takes during an episode and why we decided to use PPO instead of other deep-RL algorithms. We also introduced a novel batching algorithm for subgraph-level tasks because of the lack of batching algorithms for these tasks. Other GNN research for subgraph-level tasks could use this batching algorithm for their algorithms.

[2]https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail

Chapter 4

Experimental Setup

This chapter will discuss how we will conduct the experiments and the graphs we will use for training and to compare DCCA to previous works. We will conduct experiments for both the diversified top-k clique search problem (DTKC) and the diversified top-k weighted clique search problem (DTKWC). First, we analyse the generated graphs we used in training Deep Clique Comparison Agent (DCCA) and for the comparison, and later in the chapter, we will discuss the measurements we will use in the comparison.

We will compare DCCA on three different sizes of k, namely $k = 10$, $k = 30$ and $k = 50$. We do not test for $k = 20$ and $k = 40$ due to the scope of this thesis, and the results of the other three settings should give enough information about how DCCA reacts to different sizes of k.

As baseline, we will use *TOPKLS* (Wu et al., 2020) for DTKC and *TOPKWCLQ* (Wu and Yin, 2021b) for DTKWC. We will train DCCA for both problems and all the previously stated values of k. Lastly, we will train DCCA on different data sets to see how well DCCA can generalise and scale, and what the best method is to train it.

4.1 Graph Analysis

In our comparison, we will use generated graphs and real-world graphs to see how well DCCA can generalise. Therefore, we will analyse each graph generation function by generating 1000 graphs. The analysis will show the input parameters for each graph, the number of cliques in the graph, the mean, the maximum and minimum size of a clique and standard deviation and the mean degree.

4.1.1 Generated Graphs

We decided to use only graphs generated by the Dual Barabási–Albert (BA) model. The reason is that other models, such as the normal BA model and the Erdős-Rényi (ER) model, do not generate graphs with diverse enough clique sizes without needing to be of enormous sizes[1]. This subsection will analyse graphs generated by the Dual BA model

[1] In appendix 4.2, we show an analysis of these models.

for training our agents and evaluating them.

We generate the first set of graphs with the same input variables, namely: $n = 1500$, $m_1 = 6$, $m_2 = 150$ and $p = 0.98$[2]. Table 4.1 shows the results of these graphs. For the evaluation, we generated 10 graphs with these input variables. The sections that show the results of these graphs are called: **Dual Barabási–Albert model - Same Parameters**.

Value	Mean	SD	Max	Min
Mean Clique Size	4.592	0.923	8.61	2.921
SD Clique Size	3.183	0.903	6.014	1.277
Max Clique Size	17.523	2.481	25	10
Number of Cliques	8374.086	976.968	13783	6721
$\|V\|$	1500	0	1500	1500
$\|E\|$	12112.272	753.138	14436	9828
Mean Degree	16.15	1.004	19.248	13.104

Table 4.1: The results of the graphs generated with the same input parameters

We created graphs where each input variable was randomly chosen from a given range for the second set of graphs. n is between 1250 and 2250, m_1 is between 5 and 10, m_2 is between 100 and 150 and p between 0.95 and 0.98. Table 4.2 shows the results of the generated graphs. We chose to generate 15 graphs for the evaluation because of the increased randomness. The sections that show the results of these graphs are called: **Dual Barabási–Albert model - Random Parameters**

Value	Mean	SD	Max	Min
Mean Clique Size	7.399	2.94	19.176	3.111
SD Clique Size	3.816	1.1	7.482	1.237
Max Clique Size	17.932	3.927	32	9
Number of Cliques	30463.952	26255.725	303074	5394
$\|V\|$	1741.541	284.468	2250	1250
$\|E\|$	18966.346	4733.997	33235	8155
Mean Degree	21.745	3.88	31.949	12.345

Table 4.2: The results for the mixed set of graphs.

Table 4.3 shows the analysis of the graphs generated with $n = 1250$, $m_1 = 5$, $m_2 = 100$ and $p = 0.98$, which are the least complex graphs possible to generate.

[2]We explained the function of each parameter in section 2.1

Value	Mean	SD	Max	Min
Mean Clique Size	3.592	0.506	6.298	2.623
SD Clique Size	2.078	0.526	3.991	0.913
Max Clique Size	12.555	1.847	18	8
Number of Cliques	5906.676	514.628	9324	4939
$\|V\|$	1250	0	1250	1250
$\|E\|$	8028.29	444.816	9550	6795
Mean Degree	12.845	0.712	15.28	10.872

Table 4.3: The results of the least complex graphs possible in the mixed set

Table 4.4 shows the analysis of the graphs generated with $n = 2250$, $m_1 = 10$, $m_2 = 150$ and $p = 0.95$, which are the most complex graphs possible to generate.

Value	Mean	SD	Max	Min
Mean Clique Size	10.997	1.18	15.154	7.011
SD Clique Size	4.346	0.429	5.928	3.152
Max Clique Size	22.631	2.117	31	17
Number of Cliques	129840.406	44073.95	360661	42258
$\|V\|$	2250	0	2250	2250
$\|E\|$	35823.34	1341.627	40040	31360
Mean Degree	31.843	1.193	35.591	27.876

Table 4.4: The results of the most complex graphs possible in the mixed set

4.1.2 Real-world Graphs

We selected graphs from the same set as *TOPKLS* (Wu et al., 2020) for the real-world graphs (Rossi and Ahmed, 2015). We chose 15 random graphs from this data set, from the in total of 139 graphs, with the maximum size being 200,000 nodes. The reason why we only chose 15 is that the scope of our thesis is too small to test more graphs. Non of these graphs are weighted; thus, we add a random weight to each node between 1 and 10. Wu and Yin (2021b) does a similar method to all add weights to non-weighted graphs, to test *TOPKWCLQ*. In section 4.2, we show an analysis of each of those graphs and the other graphs used to evaluate our agents.

4.2 Evaluation Graphs

This section gives an overview of the graphs we used in the evaluation. The tables show the number of nodes and edges of a given graph, with the total number of cliques, with the mean, max and standard deviation size of those cliques found.

4.2.1 Dual Barabási–Albert model - Same Parameters

Table 4.5 shows the graphs we will use for the evaluation, which we generated using the same input parameters found in table 4.1. The measured attributes show that these graphs are all similar.

				Clique Size		
Graph	$\|V\|$	$\|E\|$	**Total Maximal Cliques**	**Mean**	**SD**	**Max**
graph_0	1500	12276	8391	4.376	2.95	18
graph_1	1500	11556	8425	3.931	2.41	14
graph_2	1500	11844	7772	4.252	2.871	17
graph_3	1500	12132	7660	4.499	3.411	19
graph_4	1500	11844	8046	4.153	2.767	18
graph_5	1500	12708	7998	4.59	3.44	21
graph_6	1500	11412	7949	3.624	2.08	15
graph_7	1500	11124	7758	3.617	2.128	15
graph_8	1500	11412	7530	3.781	2.316	15
graph_9	1500	12420	8896	5.387	3.778	17

Table 4.5: The evaluation graphs generated with same parameters as the ones found in table 4.1, namely, $n = 1500$, $m_1 = 6$, $m_2 = 150$ and $p = 0.98$. The first column ($|V|$) shows the number of nodes in the graph and the second column ($|E|$) the number of edges. In the third column, we show the total number of maximal cliques in that graph, with the last three being the mean, standard deviation (SD) and maximum size of those found cliques.

4.2.2 Dual Barabási–Albert model - Random Parameters

Table 4.6 shows the graph we generated from the random range input attributes. In it, we see that the measured attributes of these graphs vary significantly more than those found in table 4.5. We expected this variation because of the random input parameters. However, the most important difference is the total number of cliques, which is, on average, significantly larger than the graphs found in table 4.5.

Graph	$\|V\|$	$\|E\|$	Total Maximal Cliques	Clique Size		
				Mean	SD	Max
graph_0	2011	19102	30305	8.872	4.767	19
graph_1	1305	15521	20476	5.802	2.726	14
graph_2	1941	18063	36603	8.306	3.693	19
graph_3	2123	16398	18817	8.176	5.338	18
graph_4	1958	22833	176203	14.458	4.124	25
graph_5	1609	18066	58995	11.824	4.268	22
graph_6	2175	26158	31070	4.434	2.114	12
graph_7	1376	15268	14466	5.099	2.898	15
graph_8	2185	21878	42139	9.961	4.751	18
graph_9	1382	12813	26959	11.093	4.8	21
graph_10	1817	20744	21678	7.305	4.865	20
graph_11	2072	25336	69770	10.024	4.108	21
graph_12	1984	18827	17366	4.998	3.167	15
graph_13	1277	14568	43274	20.944	7.045	32
graph_14	1337	11178	8174	5.334	3.865	18

Table 4.6: The evaluation graphs generated with same parameters as the ones found in table 4.2, namely, with n being between 1250 and 2250, m_1 being between 5 and 10, m_2 being between 100 and 150 and p between 0.95 and 0.98

4.2.3 Real-world Graphs

The last set of evaluation graphs is the set of real-world graphs. Table 4.7 shows that these graphs vary the most in their measured attributes. For example, *rt-retweet* only has 99 maximal cliques, while *soc-buzznet* has 390,020,762 maximal cliques. This size likely indicates that DCCA needs days to check all the cliques in *soc-buzznet*, which would mean it is drastically slower than previous methods, such as *TOPKLS*.

Graph	$\|V\|$	$\|E\|$	Total Maximal Cliques	Clique Size		
				Mean	SD	Max
ca-GrQc	5242	14484	3906	3.053	2.012	44
ca-netscience	379	914	203	3.557	1.244	9
ia-email-univ	1133	5451	3267	3.035	1.096	12
ia-infect-dublin	410	2765	1247	5.018	2.452	16
ia-infect-hyper	113	2196	5347	9.423	2.29	15
inf-power	4941	6594	5687	2.092	0.326	6
rt-retweet	96	117	99	2.101	0.333	4
sc-shipsec1	140385	1707759	505189	5.775	2.654	24
soc-buzznet	101163	2763066	390020762	15.317	4.113	31
socfb-CMU	6621	249959	1242538	12.291	6.169	45
tech-RL-caida	190914	607610	480784	2.893	1.501	17
tech-WHOIS	7476	56943	3575771	42.985	7.498	58
tech-internet-as	40164	85123	69584	2.919	1.633	16
tech-routers-rf	2113	6632	3457	3.219	1.912	16
web-arabic-2005	163598	1747269	93444	3.326	6.01	102
web-spam	4767	37375	64228	8.078	3.616	20

Table 4.7: This table shows the real-world graphs we use for our evaluation.

4.3 Trained Agents

This section will explain each trained version of Deep Clique Comparison Agent (DCCA) and show for which problem they were trained and which data set they used. As mentioned in the introduction of this chapter, we will only train the agents for $k = 10$, $k = 30$, and $k = 50$. We trained these agents on the data sets discussed in the graph analysis section. DCCA trained on the graphs in table 4.1, we notate with **DCCA-same** and when trained on the graphs in table 4.2 we notate it as **DCCA-mix**. To train them for DTKWC, we add random weights to each node between 1 and 10. We will compare these versions of DCCA in our results to determine the effect of the training graphs for the results. We also trained DCCA for both the diversified top-k clique search problem and the diversified top-k weighted clique search problem.

4.3.1 Hyperparameters

Table 4.8 shows the hyperparameters for DCCA, the only distinction between the weighted version and the non-weighted version is the lower reward scaling. We used the same network design as Abe et al. (2019), which means we used 5 GIN layers in the encoder; each MLP in the GIN has an input size and output size of 32 and a hidden size of 16 for each hidden layer, with three hidden layers. The actor and critic networks also use GIN with the same MLP design, except that the output size is 1. The input size of the first GIN layer is 33 for DTKWC, because we add the normalised weight to each nodes input. We used ReLU for the activation function (Agarap, 2018). We used Generalised

Advantage Estimation (GAE) to calculate the advantage during training. We added a cutoff time of 900 seconds to DCCA. Otherwise, DCCA needs days to run on larger graphs.

We trained DCCA for 4 million steps. After each 100 episodes, we compared DCCA to some test cases and saved those results and the weights of the networks. In the final comparison, we used the weights that scored the best at those test cases. Our reasoning behind this is that we train DCCA for a significant amount of different setups; therefore, the ideal number of steps varies for values of k and if it is either trained for DTKC or DTKWC.

Hyperparameter	**Value**
Discount Factor γ	0.99
GAE-lambda λ	0.9
Learning Rate α	2.5×10^{-4}
Epochs E	5
Batch Size B	128
PPO Horizon H	4096
Clip Rate ϵ	0.1
Value Coefficient c_1	0.5
Entropy Coefficient c_2	0.01
Reward Scaling DTKC ρ	0.1
Reward Scaling DTKWC ρ	0.01

Table 4.8: We tuned the hyperparameters based on the non-weighted variant for $k = 10$

For *TOPKLS* (DTKC) and *TOPKWCLQ* (DWTKC), we used the input parameters of *TOPKLS* (Wu et al., 2020). Therefore, we set $m_{\min} = 50$ and $m_{\max} = 200$, and we tested for a cutoff time of 600 seconds and 60 seconds.

4.4 Interpreting Results

As mentioned in the introduction of this chapter, we will compare DCCA, trained with hyperparameters in table 4.8, to *TOPKLS* and *TOPKWCLQ*. Due to the stochastic nature of both *TOPKLS* and *TOPKWCLQ*, we have run both multiple times with different seeds. We decided to repeat a run five times with the following seeds: 1234, 2345, 3456, 4567 and 5678. The average of these runs is then the final score. DCCA is deterministic and thus only runs once. Ideally, we would have trained DCCA multiple times and run each trained version on the evaluation graphs. However, we found too late in the process that this is common for RL algorithms and we did not have enough time to do this full comparison.

Our results will state the average score, standard deviation of this score, and the maximum score of the five runs for *TOPKLS* and *TOPKWCLQ*. The score is either the length of the coverage set for DTKC or the summation of the nodes weights in that coverage set for DTKWC. We will state the end score of a graph and the maximum score found during that run for either trained version of DCCA, namely, DCCA-Same, the

one trained on graphs with the same parameter (table 4.1) and DCCA-Mix, the one on a broader range (table 4.2). DCCA can output a lower score than the maximum score it found during their run. The difference between the output score and the maximum score can determine the stability of DCCA and will indicate future improvements. We will state the percentage difference between the average score of *TOPKLS* or *TOPKWCLQ* and the max score of DCCA.

Both the baselines, *TOPKLS* and *TOPKWCLQ*, were tested with a cutoff time of 600 seconds. We already argued that DCCA is probably faster on graphs with fewer cliques because it only checks each clique once. However, to test if this speed-up is significant, we will also test both baselines with a cutoff time of 60 seconds. We then can compare both cutoff times to conclude if our speed-up is significant. If this difference is insignificant, it would mean that the baseline algorithms find similar solutions for the lower cutoff time; therefore, our speed-up advantage is also insignificant. To test the significance, we will use a dependent T-test, in which the percentage difference between the max score of DCCA and the mean result of the baseline is compared between the two cutoff time. This test will be done for both versions of DCCA, so for DCCA-Same, trained on the dual BA graphs generated by the same input parameters (table 4.1), and DCCA-Mix, trained on the dual BA graphs from a range of input parameters (table 4.2). We will do this for each evaluation graph set and value k.

However, we can not make a fair comparison between the runtime of DCCA and that of *TOPKLS* (Wu et al., 2020) and *TOPKWCLQ* (Wu and Yin, 2021b) because DCCA is written in Python, and *TOPKLS* and *TOPKWCLQ* are in C++ and are therefore likely inherently faster. Nevertheless, different graphs, problems or values of k can affect the final run time and tell us how DCCA holds up in different scenarios.

Chapter 5

Results

In the results chapter, we will show the results of DCCA. We divided this chapter into two sections: one for the diversified top-k clique search problem (DTKC) and one for the diversified top-k weighted clique search problem (DTKWC). In each section, we will discuss the results of each data set separately and compare DCCA-Same and DCCA-Mix to either *TOPKLS* for DTKC or *TOPKWCLQ* for DTKWC.

For all the evaluation graph sets, we will show the results for each of tested value of k, which are $k = 10$, $k = 30$ and $k = 50$. We separated each table into three parts, one for *TOPKLS* or *TOPKWCLQ* and one for both DCCA-Same and DCCA-Mix. The part of *TOPKLS* and *TOPKWCLQ* shows the mean results of five runs, with the standard deviation in parentheses, in the **Result** column, with a separate column for the maximum score found in these five runs. As previously mentioned, we used cutoff times of 600 seconds and 60 seconds for *TOPKLS* and *TOPKWCLQ*.

The results for DCCA show the maximum found score during evaluation, the end score of the clique set, which DCCA returned at the end of the run and the total run time. The **End Score** column also shows the percentage difference between the end and the maximum scores in parentheses. In the **Max Score** column, we also state the percentage difference between the maximum score and the mean result of either *TOPKLS* or *TOPKWCLQ*. If the run time is written in bold, it means that DCCA terminated before it checked all the cliques because it exceeded the cutoff time of 900 seconds. Appendix C shows the full results of DCCA, such as the total number of steps taken, and when the max score was found. Earlier, we said we tested for both a cutoff time of 600 seconds and 60 seconds. We use a dependent t-test to see if the difference between the results of the two cutoff times is significant. To test this significance, we compare the percentage difference between the max score of DCCA and the results either of *TOPKLS* or *TOPKWCLQ* for a cutoff of 600 seconds and 60 seconds. We do this for both DCCA-Mix and DCCA-Same. For these tests, the significance level is $\alpha = 0.01$.

In each table, we state the mean difference with standard deviation between the maximum score of DCCA and the mean result of the baseline algorithm and between the maximum and end score. In each table, we will show the best result in bold. When we discuss the results of the real-world graphs, we will show separately the mean difference between the graphs which DCCA completed and the ones it terminated early on.

5.1 Diversified Top-k Clique Search

This section shows the results of our experiments for the diversified top-k clique search problem (DTKC). These results show that DCCA is significantly worse than the baseline, *TOPKLS*, for both cutoff times. When the cutoff time is 600 seconds, the average percentage difference between the max score of DCCA and the mean result of *TOPKLS* is for DCCA-Same -23.02% (SD=14.07%) and DCCA-Mix it is -26.62% (SD=14.24%). If we lower the cutoff time to 60 seconds, we see the difference being lower between the max score and the mean result. However, it is still considerably worse, with the difference for DCCA-Same being -22.14% (SD=14.16%) and for DCCA-Mix being -25.78% (SD=14.38%). Nevertheless, the results vary significantly between the graph sets and for different values of k. For example, DCCA-Mix outperformed *TOPKLS* on average on the dual BA graphs generated with the same input parameters (table 4.5) for both a cutoff time of 600 seconds (table 5.3) and 60 seconds (table 5.6). While DCCA-Same also outperformed on the same graph set, with $k = 50$, but only when the cutoff is 60 seconds (table 5.6).

Moreover, DCCA is also significantly faster on those graphs. This indicates that DCCA is an improvement on both runtime and score for those graphs and when $k = 50$. However, on the largest graphs found in the results of the real-world graphs (section 5.1.3), DCCA terminated the run because it reached its cutoff time of 900 seconds, indicating that it can not scale to those graphs. This is because there are too many maximal cliques in those graphs. We will expand on this in our discussion.

5.1.1 Dual Barabási–Albert model - Same Parameters

Tables 5.1, 5.2 and 5.3 show the results of the graphs generated with the same input parameters. We compared *TOPKLS* to both DCCA-Same, which is trained on dual BA graphs generated with the same input parameters, similar to how these graphs are generated and the ones in section 5.2.1, and DCCA-Mix, which is trained on dual BA graphs from a range of input variables, similar to the graph found in section 5.1.2 and 5.2.2. In the results of this section, we see, on average, *TOPKLS* outperforming DCCA-Mix and DCCA-Same for $k = 10$ and $k = 30$. However, the difference in performance is smaller for $k = 30$ than it is for $k = 10$. Moreover, DCCA-Mix even outperforms *TOPKLS* when $k = 50$, and DCCA-Same is only slightly worse than *TOPKLS* with a cutoff time of 600. When the cutoff time is 60, then DCCA-Same also outperforms *TOPKLS* for $k = 50$. When we compare DCCA-Mix and DCCA-Same, we see that DCCA-Same performs better when $k = 10$ and $k = 30$, and DCCA-Mix is better when $k = 50$.

TOPKLS cutoff of 600 seconds

When we combine the results of all the three tested values of k (tables 5.1, 5.2 and 5.3), we see that the mean difference between *TOPKLS* and DCCA-Same is -4.25% (SD = 3.25) and the mean difference between *TOPKLS* and DCCA-Mix is -16.94% (SD = 4.29%). The mean difference between the end and the max score for DCCA-Same is -3.43% (SD = 4.29%) and for DCCA-Mix is -11.68% (SD = 5.76%). This indicates

that on average DCCA-Same is better compared to DCCA-Mix. However, DCCA-Mix outperformed *TOPKLS* when $k = 50$ (table 5.3).

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**51.2 (0.4)**	52	600	49 (-2.0%)	50 (-2.34%)	6.8	24 (-33.33%)	36 (-29.69%)	6.83
graph_1	**48.4 (0.49)**	49	600	41 (-8.89%)	45 (-7.02%)	6.85	18 (-41.94%)	31 (-35.95%)	6.78
graph_2	**49.8 (0.4)**	50	600	44 (0.0%)	44 (-11.65%)	6.2	23 (-36.11%)	36 (-27.71%)	6.25
graph_3	**52.0 (0.0)**	52	600	46 (-2.13%)	47 (-9.62%)	6.23	25 (-24.24%)	33 (-36.54%)	6.23
graph_4	**50.2 (0.4)**	51	600	42 (-12.5%)	48 (-4.38%)	6.56	22 (-31.25%)	32 (-36.25%)	6.46
graph_5	**54.2 (0.4)**	55	600	48 (-4.0%)	50 (-7.75%)	6.5	29 (-19.44%)	36 (-33.58%)	6.59
graph_6	**47.0 (0.0)**	47	600	45 (0.0%)	45 (-4.26%)	6.41	19 (-36.67%)	30 (-36.17%)	6.36
graph_7	**46.4 (0.49)**	47	600	42 (0.0%)	42 (-9.48%)	6.45	21 (-25.0%)	28 (-39.66%)	6.4
graph_8	46.8 (0.4)	47	600	**47 (0.0%)**	**47 (0.43%)**	6.21	23 (-28.12%)	32 (-31.62%)	6.03
graph_9	**52.4 (0.49)**	53	600	39 (-13.33%)	45 (-14.12%)	7.2	25 (-21.88%)	32 (-38.93%)	7.21

Table 5.1: The results for $k = 10$. The columns for *TOPKLS* show the mean results, with standard deviation, in the **Result** column from the 5 runs, while the **Max** column shows highest found score of those 5 runs. In the results for DCCA-Same and DCCA-Mix, we show the score of the final clique set with the percentual difference between it and the max score in the **End Score** column. In the **Max Score** column, the highest found score during the run with percentual difference between it and the mean result of *TOPKLS*. **DCCA-Same**: mean difference between TOPKLS is -7.02% (SD = 4.44%) with the mean difference between the end and the max score being -4.28% (SD = 5.31%). **DCCA-Mix**: mean difference between TOPKLS is -34.61% (SD = 3.9%) with the mean difference between the end and the max score being -29.8% (SD = 7.27%).

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**110.0 (0.63)**	111	600	101 (-4.72%)	106 (-3.64%)	10.52	88 (-1.12%)	89 (-19.09%)	9.81
graph_1	**108.0 (0.0)**	108	600	96 (-3.03%)	99 (-8.33%)	10.39	62 (-24.39%)	82 (-24.07%)	9.91
graph_2	**108.4 (0.49)**	109	600	102 (-2.86%)	105 (-3.14%)	9.76	94 (0%)	94 (-13.28%)	9.05
graph_3	**109.8 (0.75)**	111	600	95 (-9.52%)	105 (-4.37%)	9.53	73 (-13.1%)	84 (-23.5%)	9.04
graph_4	**109.0 (0.0)**	109	600	98 (-3.92%)	102 (-6.42%)	10.01	85 (-1.16%)	86 (-21.1%)	9.37
graph_5	**113.0 (0.0)**	113	600	98 (-5.77%)	104 (-7.96%)	9.77	100 (0%)	100 (-11.5%)	9.42
graph_6	**106.4 (0.49)**	107	600	99 (-3.88%)	103 (-3.2%)	10.05	90 (0%)	90 (-15.41%)	9.26
graph_7	**105.8 (0.4)**	106	600	102 (0%)	102 (-3.59%)	9.89	91 (0%)	91 (-13.99%)	9.32
graph_8	**105.0 (0.0)**	105	600	99 (-3.88%)	103 (-1.9%)	9.39	91 (0%)	91 (-13.33%)	8.89
graph_9	**111.2 (0.4)**	112	600	95 (-4.04%)	99 (-10.97%)	11.29	100 (0%)	100 (-10.07%)	10.47

Table 5.2: The results for $k = 30$. **DCCA-Same**: mean difference between TOPKLS is -5.35% (SD = 2.92%) with the mean difference between the end and the max score being -4.16% (SD = 2.41%). **DCCA-Mix**: mean difference between TOPKLS is -16.54% (SD = 5.04%) with the mean difference between the end and the max score being -3.98% (SD = 8.24%).

Graphs	TOPKLS			DCCA-Same			DCCA-Mix		
	Result	Max	Run Time	End Score	Max Score	Run Time	End Score	Max Score	Run Time
graph_0	158.0 (1.26)	160	600	158 (-2.47%)	162 (2.53%)	13.44	163 (-1.21%)	**165 (4.43%)**	13.62
graph_1	159.8 (0.98)	161	600	157 (-3.09%)	**162 (1.38%)**	13.51	148 (-1.99%)	151 (-5.51%)	13.67
graph_2	157.4 (0.8)	159	600	155 (-2.52%)	**159 (1.02%)**	12.84	157 (0%)	157 (-0.25%)	12.6
graph_3	154.2 (0.4)	155	600	156 (-0.64%)	157 (1.82%)	12.51	153 (-5.56%)	**162 (5.06%)**	12.5
graph_4	157.6 (0.49)	158	600	150 (-1.96%)	153 (-2.92%)	12.86	**159 (0%)**	**159 (0.89%)**	13.03
graph_5	**160.8 (1.47)**	163	600	152 (-3.18%)	157 (-2.36%)	12.99	158 (0%)	158 (-1.74%)	12.93
graph_6	**157.2 (0.75)**	158	600	150 (-1.96%)	153 (-2.67%)	12.7	151 (-1.95%)	154 (-2.04%)	12.85
graph_7	155.8 (0.4)	156	600	153 (-0.65%)	154 (-1.16%)	12.48	**161 (0%)**	**161 (3.34%)**	12.35
graph_8	151.4 (0.8)	152	600	152 (-1.94%)	155 (2.38%)	12.08	**158 (0%)**	**158 (4.36%)**	12.16
graph_9	**159.4 (1.02)**	161	600	154 (0%)	154 (-3.39%)	14.45	148 (-1.99%)	151 (-5.27%)	14.37

Table 5.3: The results for $k = 50$. **DCCA-Same**: mean difference between TOPKLS is -0.34% (SD = 2.39%) with the mean difference between the end and the max score being -1.84% (SD = 1.08%). **DCCA-Mix**: mean difference between TOPKLS is 0.33% (SD = 3.95%) with the mean difference between the end and the max score being -1.27% (SD = 1.76%).

TOPKLS cutoff of 60 seconds

The combination of the the results of all the three tested values of k (tables 5.4, 5.5, and 5.6) with a cutoff of 60 seconds, show that the mean difference between *TOPKLS* and DCCA-Same is -3.14% (SD = 3.35) and the mean difference between *TOPKLS* and DCCA-Mix is -16% (SD = 4.42%).

Graphs	TOPKLS			DCCA-Same			DCCA-Mix		
	Result	Max	Run Time	End Score	Max Score	Run Time	End Score	Max Score	Run Time
graph_0	**50.2 (0.4)**	51	60	49 (-2.0%)	50 (-0.4%)	6.8	24 (-33.33%)	36 (-28.29%)	6.83
graph_1	**47.6 (0.49)**	48	60	41 (-8.89%)	45 (-5.46%)	6.85	18 (-41.94%)	31 (-34.87%)	6.78
graph_2	**49.0 (0.63)**	50	60	44 (0.0%)	44 (-10.2%)	6.2	23 (-36.11%)	36 (-26.53%)	6.25
graph_3	**51.0 (0.0)**	51	60	46 (-2.13%)	47 (-7.84%)	6.23	25 (-24.24%)	33 (-35.29%)	6.23
graph_4	**49.2 (0.4)**	50	60	42 (-12.5%)	48 (-2.44%)	6.56	22 (-31.25%)	32 (-34.96%)	6.46
graph_5	**53.4 (0.49)**	54	60	48 (-4.0%)	50 (-6.37%)	6.5	29 (-19.44%)	36 (-32.58%)	6.59
graph_6	**46.8 (0.4)**	47	60	45 (0.0%)	45 (-3.85%)	6.41	19 (-36.67%)	30 (-35.9%)	6.36
graph_7	**46.0 (0.63)**	47	60	42 (0.0%)	42 (-8.7%)	6.45	21 (-25.0%)	28 (-39.13%)	6.4
graph_8	46.2 (0.4)	47	60	**47 (0.0%)**	**47 (1.73%)**	6.21	23 (-28.12%)	32 (-30.74%)	6.03
graph_9	**52.0 (0.63)**	53	60	39 (-13.33%)	45 (-13.46%)	7.2	25 (-21.88%)	32 (-38.46%)	7.21

Table 5.4: The results for $k = 10$. **DCCA-Same**: mean difference between TOPKLS is -5.7% (SD = 4.62%) with the mean difference between the end and the max score being -4.28% (SD = 5.31%). **DCCA-Mix**: mean difference between TOPKLS is -33.68% (SD = 4.13%) with the mean difference between the end and the max score being -29.8% (SD = 7.27%).

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**109.0 (0.0)**	109	60	101 (-4.72%)	106 (-2.75%)	10.52	88 (-1.12%)	89 (-18.35%)	9.81
graph_1	**107.2 (0.4)**	108	60	96 (-3.03%)	99 (-7.65%)	10.39	62 (-24.39%)	82 (-23.51%)	9.91
graph_2	**107.8 (0.4)**	108	60	102 (-2.86%)	105 (-2.6%)	9.76	94 (0.0%)	94 (-12.8%)	9.05
graph_3	**109.2 (0.98)**	111	60	95 (-9.52%)	105 (-3.85%)	9.53	73 (-13.1%)	84 (-23.08%)	9.04
graph_4	**108.4 (0.49)**	109	60	98 (-3.92%)	102 (-5.9%)	10.01	85 (-1.16%)	86 (-20.66%)	9.37
graph_5	**112.0 (0.0)**	112	60	98 (-5.77%)	104 (-7.14%)	9.77	100 (0.0%)	100 (-10.71%)	9.42
graph_6	**105.8 (0.75)**	107	60	99 (-3.88%)	103 (-2.65%)	10.05	90 (0.0%)	90 (-14.93%)	9.26
graph_7	**105.2 (0.4)**	106	60	102 (0.0%)	102 (-3.04%)	9.89	91 (0.0%)	91 (-13.5%)	9.32
graph_8	**104.4 (0.49)**	105	60	99 (-3.88%)	103 (-1.34%)	9.39	91 (0.0%)	91 (-12.84%)	8.89
graph_9	**110.6 (0.49)**	111	60	95 (-4.04%)	99 (-10.49%)	11.29	100 (0.0%)	100 (-9.58%)	10.47

Table 5.5: The results for $k = 30$. **DCCA-Same**: mean difference between TOPKLS is -4.74% (SD = 2.92%) with the mean difference between the end and the max score being -4.16% (SD = 2.41%). **DCCA-Mix**: mean difference between TOPKLS is -16.0% (SD = 5.06%) with the mean difference between the end and the max score being -3.98% (SD = 8.24%).

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	155.0 (0.63)	156	60	158 (-2.47%)	162 (4.52%)	13.44	163 (-1.21%)	**165 (6.45%)**	13.62
graph_1	157.4 (1.2)	159	60	157 (-3.09%)	**162 (2.92%)**	13.51	148 (-1.99%)	151 (-4.07%)	13.67
graph_2	154.8 (1.83)	157	60	155 (-2.52%)	**159 (2.71%)**	12.84	157 (0.0%)	157 (1.42%)	12.6
graph_3	152.8 (1.6)	155	60	156 (-0.64%)	157 (2.75%)	12.51	153 (-5.56%)	**162 (6.02%)**	12.5
graph_4	155.2 (0.75)	156	60	150 (-1.96%)	153 (-1.42%)	12.86	159 (0.0%)	**159 (2.45%)**	13.03
graph_5	**159.0 (1.9)**	162	60	152 (-3.18%)	157 (-1.26%)	12.99	158 (0.0%)	158 (-0.63%)	12.93
graph_6	**154.8 (0.98)**	156	60	150 (-1.96%)	153 (-1.16%)	12.7	151 (-1.95%)	154 (-0.52%)	12.85
graph_7	153.2 (0.4)	154	60	153 (-0.65%)	154 (0.52%)	12.48	**161 (0.0%)**	**161 (5.09%)**	12.35
graph_8	150.2 (0.75)	151	60	152 (-1.94%)	155 (3.2%)	12.08	**158 (0.0%)**	**158 (5.19%)**	12.16
graph_9	**158.2 (1.47)**	161	60	154 (0.0%)	154 (-2.65%)	14.45	148 (-1.99%)	151 (-4.55%)	14.37

Table 5.6: The results for $k = 50$. **DCCA-Same**: mean difference between TOPKLS is 1.01% (SD = 2.5%) with the mean difference between the end and the max score being -1.84% (SD = 1.08%). **DCCA-Mix**: mean difference between TOPKLS is 1.69% (SD = 4.06%) with the mean difference between the end and the max score being -1.27% (SD = 1.76%).

Results T-Test

The results of the dependent T-test between the percentage difference of *TOPKLS* with a cutoff time of 600 seconds and 60 seconds for both DCCA-Same and DCCA-Mix are as follow:

- If $k = 10$, then for DCCA-Same, the difference between a cutoff time of 60 seconds ($M = -5.70\%$, $SD = 4.62\%$) and 600 seconds ($M = -7.02\%$, $SD = 4.44\%$) is significant, $T(9)=7.7558$, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds ($M = -33.68\%$, $SD = 4.13\%$) or 600 seconds ($M = -34.61\%$, $SD = 3.90\%$) is significant, $T(9)=7.6271$, $p < 0.01$.
- If $k = 30$, then for DCCA-Same, the difference between a cutoff time of 60 seconds ($M = -4.74\%$, $SD = 2.92\%$) and 600 seconds ($M = -5.35\%$, $SD =$

2.92%) is significant, T(9)=14.0085, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = −16.00%, SD = 5.06%) or 600 seconds (M = −16.54%, SD = 5.04%) is significant, T(9)=13.5332, $p < 0.01$.

- If $k = 50$, then for DCCA-Same, the difference between a cutoff time of 60 seconds (M = 1.01%, SD = 2.50%) and 600 seconds (M = −0.34%, SD = 2.39%) is significant, T(9)=10.0826, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = 1.69%, SD = 4.06%) or 600 seconds (M = 0.33%, SD = 3.95%) is significant, T(9)=9.9785, $p < 0.01$.

5.1.2 Dual Barabási–Albert model - Random Parameters

The results shown in tables 5.7, 5.8, and 5.9 show that *TOPKLS* outperforms DCCA on every graph, for every value of k. When we compare these results to those of the graphs generated with the input parameter (tables 5.1, 5.2 and 5.3), we see that both DCCA-Same and DCCA-Mix score worse compared to the baseline. The runtime for both is also higher for both. This likely indicates that DCCA struggles with graphs with more cliques. Nevertheless, when we compare DCCA-Mix and DCCA-Same, we see that DCCA-Same is better when $k = 10$ and $k = 30$ and DCCA-Mix is better when $k = 50$. This behaviour happened also in the previous results in section 5.1.1.

TOPKLS cutoff of 600 seconds

When we combine the results of all the three tested values of k (tables 5.7, 5.8, and 5.9), we see that the mean difference between *TOPKLS* and DCCA-Same is −21.87% (SD = 12.55) and the mean difference between *TOPKLS* and DCCA-Mix is −20.59% (SD = 10.58%). The mean difference between the end and the max score for DCCA-Same is −10.37% (SD = 8.36%) and for DCCA-Mix is −9.27% (SD = 5.64%).

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**68.6 (0.8)**	70	600	32 (-42.86%)	56 (-18.37%)	24.02	41 (-22.64%)	53 (-22.74%)	24.53
graph_1	**60.0 (0.0)**	60	600	37 (-24.49%)	49 (-18.33%)	16.41	41 (-19.61%)	51 (-15.0%)	16.04
graph_2	**70.0 (0.63)**	71	600	33 (-40.0%)	55 (-21.43%)	29.57	48 (-12.73%)	55 (-21.43%)	29.29
graph_3	**60.8 (0.75)**	62	600	36 (-32.08%)	53 (-12.83%)	15.05	32 (-25.58%)	43 (-29.28%)	15.05
graph_4	**85.2 (1.17)**	87	600	48 (-5.88%)	51 (-40.14%)	144.21	53 (-17.19%)	64 (-24.88%)	143.41
graph_5	**74.0 (0.0)**	74	600	36 (-37.93%)	58 (-21.62%)	47.18	44 (-15.38%)	52 (-29.73%)	47.25
graph_6	**59.6 (0.8)**	61	600	37 (-17.78%)	45 (-24.5%)	24.76	46 (-8.0%)	50 (-16.11%)	24.8
graph_7	**57.4 (0.49)**	58	600	46 (-8.0%)	50 (-12.89%)	11.49	32 (-25.58%)	43 (-25.09%)	11.49
graph_8	**68.8 (0.4)**	69	600	33 (-35.29%)	51 (-25.87%)	33.55	44 (-15.38%)	52 (-24.42%)	33.85
graph_9	**69.4 (0.49)**	70	600	49 (-3.92%)	51 (-26.51%)	21.53	30 (-21.05%)	38 (-45.24%)	21.64
graph_10	**65.4 (0.49)**	66	600	40 (-23.08%)	52 (-20.49%)	17.53	49 (-15.52%)	58 (-11.31%)	17.39
graph_11	**74.8 (1.17)**	77	600	34 (-35.85%)	53 (-29.14%)	56.61	55 (-3.51%)	57 (-23.8%)	55.66
graph_12	**57.0 (0.63)**	58	600	43 (-2.27%)	44 (-22.81%)	13.68	43 (-8.51%)	47 (-17.54%)	13.78
graph_13	**77.8 (0.4)**	78	600	41 (-16.33%)	49 (-37.02%)	35.35	43 (-12.24%)	49 (-37.02%)	35.52
graph_14	**54.6 (0.49)**	55	600	46 (0.0%)	46 (-15.75%)	6.49	28 (-24.32%)	37 (-32.23%)	6.43

Table 5.7: The results for $k = 10$. **DCCA-Same**: mean difference between TOPKLS is -23.18% (SD = 7.85%) with the mean difference between the end and the max score being -21.72% (SD = 15.11%). **DCCA-Mix**: mean difference between TOPKLS is -25.05% (SD = 8.82%) with the mean difference between the end and the max score being -16.48% (SD = 6.74%).

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**130.2 (0.75)**	131	600	91 (-1.09%)	92 (-29.34%)	36.27	109 (-4.39%)	114 (-12.44%)	36.27
graph_1	**123.6 (0.49)**	124	600	99 (-12.39%)	113 (-8.58%)	24.46	83 (-14.43%)	97 (-21.52%)	24.22
graph_2	**133.2 (0.4)**	134	600	90 (-2.17%)	92 (-30.93%)	43.86	115 (-0.86%)	116 (-12.91%)	43.66
graph_3	**118.8 (0.75)**	120	600	97 (0.0%)	97 (-18.35%)	22.37	111 (0.0%)	111 (-6.57%)	22.44
graph_4	**149.6 (1.02)**	151	600	75 (-9.64%)	83 (-44.52%)	213.13	102 (-0.97%)	103 (-31.15%)	209.97
graph_5	**135.4 (0.8)**	136	600	95 (-4.04%)	99 (-26.88%)	69.01	115 (-4.17%)	120 (-11.37%)	70.37
graph_6	**125.8 (0.75)**	127	600	89 (-8.25%)	97 (-22.89%)	37.31	99 (-9.17%)	109 (-13.35%)	36.94
graph_7	**118.8 (0.4)**	119	600	100 (-2.91%)	103 (-13.3%)	17.65	69 (-29.59%)	98 (-17.51%)	16.98
graph_8	**129.2 (0.4)**	130	600	94 (-5.05%)	99 (-23.37%)	50.99	98 (-10.09%)	109 (-15.63%)	50.08
graph_9	**127.8 (0.4)**	128	600	108 (-4.42%)	113 (-11.58%)	32.25	50 (-16.67%)	60 (-53.05%)	32.7
graph_10	**126.6 (1.02)**	128	600	98 (-3.92%)	102 (-19.43%)	25.41	106 (-7.02%)	114 (-9.95%)	25.3
graph_11	**141.2 (0.98)**	143	600	91 (-4.21%)	95 (-32.72%)	89.02	104 (-8.77%)	114 (-19.26%)	82.2
graph_12	**117.6 (0.8)**	119	600	97 (-6.73%)	104 (-11.56%)	20.66	102 (-1.92%)	104 (-11.56%)	20.47
graph_13	**130.6 (0.49)**	131	600	109 (0.0%)	109 (-16.54%)	53.98	52 (-11.86%)	59 (-54.82%)	53.99
graph_14	**113.4 (0.49)**	114	600	109 (0.0%)	109 (-3.88%)	9.69	53 (-14.52%)	62 (-45.33%)	9.81

Table 5.8: The results for $k = 30$. **DCCA-Same**: mean difference between TOPKLS is -20.93% (SD = 10.74%) with the mean difference between the end and the max score being -4.32% (SD = 3.67%). **DCCA-Mix**: mean difference between TOPKLS is -22.43% (SD = 16.01%) with the mean difference between the end and the max score being -8.96% (SD = 7.85%).

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**184.8 (0.98)**	186	600	151 (-1.31%)	153 (-17.21%)	49.7	143 (-3.38%)	148 (-19.91%)	47.61
graph_1	**183.6 (0.49)**	184	600	154 (-3.14%)	159 (-13.4%)	32.93	158 (-4.24%)	165 (-10.13%)	31.44
graph_2	**187.2 (0.75)**	188	600	154 (-3.14%)	159 (-15.06%)	60.71	143 (-7.74%)	155 (-17.2%)	57.22
graph_3	**163.6 (1.02)**	165	600	153 (0.0%)	153 (-6.48%)	31.09	153 (-1.29%)	155 (-5.26%)	29.35
graph_4	**202.0 (0.0)**	202	600	58 (-22.67%)	75 (-62.87%)	304.08	136 (-2.86%)	140 (-30.69%)	280.6
graph_5	**188.6 (1.02)**	190	600	162 (-1.22%)	164 (-13.04%)	96.15	157 (-0.63%)	158 (-16.22%)	92.27
graph_6	**185.2 (0.75)**	186	600	141 (-4.73%)	148 (-20.09%)	49.86	143 (-4.03%)	149 (-19.55%)	48.33
graph_7	**178.4 (0.49)**	179	600	149 (-5.7%)	158 (-11.43%)	24.01	154 (0.0%)	154 (-13.68%)	23.24
graph_8	**183.0 (0.63)**	184	600	160 (-0.62%)	161 (-12.02%)	69.12	151 (-0.66%)	152 (-16.94%)	65.48
graph_9	**173.4 (0.49)**	174	600	95 (0.0%)	95 (-45.21%)	46.11	159 (-0.62%)	160 (-7.73%)	43.17
graph_10	**184.8 (0.4)**	185	600	152 (-2.56%)	156 (-15.58%)	35.09	159 (0.0%)	159 (-13.96%)	33.84
graph_11	**199.2 (0.4)**	200	600	158 (-8.14%)	172 (-13.65%)	115.55	152 (-6.17%)	162 (-18.67%)	111.51
graph_12	**175.4 (0.49)**	176	600	152 (-1.3%)	154 (-12.2%)	27.93	155 (-1.9%)	158 (-9.92%)	26.94
graph_13	**171.0 (0.63)**	172	600	55 (-15.38%)	65 (-61.99%)	75.29	150 (-0.66%)	151 (-11.7%)	70.19
graph_14	**162.6 (1.02)**	164	600	149 (-6.29%)	159 (-2.21%)	13.14	156 (-1.27%)	158 (-2.83%)	12.73

Table 5.9: The results for $k = 50$. **DCCA-Same**: mean difference between TOPKLS is -21.5% (SD = 19.06%) with the mean difference between the end and the max score being -5.08% (SD = 6.3%). **DCCA-Mix**: mean difference between TOPKLS is -14.29% (SD = 6.9%) with the mean difference between the end and the max score being -2.36% (SD = 2.34%).

TOPKLS cutoff of 60 seconds

When we combine the results of all the three tested values of k (tables 5.10, 5.11, and 5.12) with a cutoff of 60 seconds, we see that the mean difference between *TOPKLS* and DCCA-Same is -20.82% (SD = 12.73) and the mean difference between *TOPKLS* and DCCA-Mix is -19.52% (SD = 10.9%).

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**66.8 (0.75)**	68	60	32 (-42.86%)	56 (-16.17%)	24.02	41 (-22.64%)	53 (-20.66%)	24.53
graph_1	**59.0 (0.0)**	59	60	37 (-24.49%)	49 (-16.95%)	16.41	41 (-19.61%)	51 (-13.56%)	16.04
graph_2	**68.6 (0.8)**	70	60	33 (-40.0%)	55 (-19.83%)	29.57	48 (-12.73%)	55 (-19.83%)	29.29
graph_3	**60.0 (0.89)**	61	60	36 (-32.08%)	53 (-11.67%)	15.05	32 (-25.58%)	43 (-28.33%)	15.05
graph_4	**83.6 (0.8)**	85	60	48 (-5.88%)	51 (-39.0%)	144.21	53 (-17.19%)	64 (-23.44%)	143.41
graph_5	**72.4 (0.8)**	73	60	36 (-37.93%)	58 (-19.89%)	47.18	44 (-15.38%)	52 (-28.18%)	47.25
graph_6	**58.4 (1.74)**	61	60	37 (-17.78%)	45 (-22.95%)	24.76	46 (-8.0%)	50 (-14.38%)	24.8
graph_7	**56.2 (0.4)**	57	60	46 (-8.0%)	50 (-11.03%)	11.49	32 (-25.58%)	43 (-23.49%)	11.49
graph_8	**67.4 (1.02)**	69	60	33 (-35.29%)	51 (-24.33%)	33.55	44 (-15.38%)	52 (-22.85%)	33.85
graph_9	**68.6 (0.49)**	69	60	49 (-3.92%)	51 (-25.66%)	21.53	30 (-21.05%)	38 (-44.61%)	21.64
graph_10	**63.4 (0.49)**	64	60	40 (-23.08%)	52 (-17.98%)	17.53	49 (-15.52%)	58 (-8.52%)	17.39
graph_11	**73.0 (0.89)**	74	60	34 (-35.85%)	53 (-27.4%)	56.61	55 (-3.51%)	57 (-21.92%)	55.66
graph_12	**56.0 (1.1)**	58	60	43 (-2.27%)	44 (-21.43%)	13.68	43 (-8.51%)	47 (-16.07%)	13.78
graph_13	**77.4 (0.49)**	78	60	41 (-16.33%)	49 (-36.69%)	35.35	43 (-12.24%)	49 (-36.69%)	35.52
graph_14	**53.6 (0.8)**	55	60	46 (0.0%)	46 (-14.18%)	6.49	28 (-24.32%)	37 (-30.97%)	6.43

Table 5.10: The results for $k = 10$. **DCCA-Same**: mean difference between TOPKLS is -21.68% (SD = 8.13%) with the mean difference between the end and the max score being -21.72% (SD = 15.11%). **DCCA-Mix**: mean difference between TOPKLS is -23.57% (SD = 9.28%) with the mean difference between the end and the max score being -16.48% (SD = 6.74%).

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**127.8 (0.75)**	129	60	91 (-1.09%)	92 (-28.01%)	36.27	109 (-4.39%)	114 (-10.8%)	36.27
graph_1	**122.0 (0.0)**	122	60	99 (-12.39%)	113 (-7.38%)	24.46	83 (-14.43%)	97 (-20.49%)	24.22
graph_2	**132.0 (0.63)**	133	60	90 (-2.17%)	92 (-30.3%)	43.86	115 (-0.86%)	116 (-12.12%)	43.66
graph_3	**117.2 (0.4)**	118	60	97 (0.0%)	97 (-17.24%)	22.37	111 (0.0%)	111 (-5.29%)	22.44
graph_4	**147.8 (1.33)**	150	60	75 (-9.64%)	83 (-43.84%)	213.13	102 (-0.97%)	103 (-30.31%)	209.97
graph_5	**134.2 (0.75)**	135	60	95 (-4.04%)	99 (-26.23%)	69.01	115 (-4.17%)	120 (-10.58%)	70.37
graph_6	**124.4 (0.8)**	125	60	89 (-8.25%)	97 (-22.03%)	37.31	99 (-9.17%)	109 (-12.38%)	36.94
graph_7	**117.8 (0.4)**	118	60	100 (-2.91%)	103 (-12.56%)	17.65	69 (-29.59%)	98 (-16.81%)	16.98
graph_8	**127.6 (0.49)**	128	60	94 (-5.05%)	99 (-22.41%)	50.99	98 (-10.09%)	109 (-14.58%)	50.08
graph_9	**126.8 (0.75)**	128	60	108 (-4.42%)	113 (-10.88%)	32.25	50 (-16.67%)	60 (-52.68%)	32.7
graph_10	**124.4 (0.8)**	125	60	98 (-3.92%)	102 (-18.01%)	25.41	106 (-7.02%)	114 (-8.36%)	25.3
graph_11	**138.8 (1.17)**	141	60	91 (-4.21%)	95 (-31.56%)	89.02	104 (-8.77%)	114 (-17.87%)	82.2
graph_12	**116.6 (1.62)**	119	60	97 (-6.73%)	104 (-10.81%)	20.66	102 (-1.92%)	104 (-10.81%)	20.47
graph_13	**129.2 (0.75)**	130	60	109 (0.0%)	109 (-15.63%)	53.98	52 (-11.86%)	59 (-54.33%)	53.99
graph_14	**112.2 (0.4)**	113	60	109 (0.0%)	109 (-2.85%)	9.69	53 (-14.52%)	62 (-44.74%)	9.81

Table 5.11: The results for $k = 30$. **DCCA-Same**: mean difference between TOPKLS is -19.98% (SD = 10.78%) with the mean difference between the end and the max score being -4.32% (SD = 3.67%). **DCCA-Mix**: mean difference between TOPKLS is -21.48% (SD = 16.27%) with the mean difference between the end and the max score being -8.96% (SD = 7.85%).

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**182.8 (0.75)**	184	60	151 (-1.31%)	153 (-16.3%)	49.7	143 (-3.38%)	148 (-19.04%)	47.61
graph_1	**181.2 (0.4)**	182	60	154 (-3.14%)	159 (-12.25%)	32.93	158 (-4.24%)	165 (-8.94%)	31.44
graph_2	**186.4 (1.02)**	188	60	154 (-3.14%)	159 (-14.7%)	60.71	143 (-7.74%)	155 (-16.85%)	57.22
graph_3	**161.6 (0.8)**	163	60	153 (0.0%)	153 (-5.32%)	31.09	153 (-1.29%)	155 (-4.08%)	29.35
graph_4	**201.0 (0.89)**	202	60	58 (-22.67%)	75 (-62.69%)	304.08	136 (-2.86%)	140 (-30.35%)	280.6
graph_5	**186.2 (0.4)**	187	60	162 (-1.22%)	164 (-11.92%)	96.15	157 (-0.63%)	158 (-15.15%)	92.27
graph_6	**184.4 (0.8)**	185	60	141 (-4.73%)	148 (-19.74%)	49.86	143 (-4.03%)	149 (-19.2%)	48.33
graph_7	**177.0 (0.63)**	178	60	149 (-5.7%)	158 (-10.73%)	24.01	154 (0.0%)	154 (-12.99%)	23.24
graph_8	**181.4 (1.02)**	183	60	160 (-0.62%)	161 (-11.25%)	69.12	151 (-0.66%)	152 (-16.21%)	65.48
graph_9	**171.2 (0.98)**	172	60	95 (0.0%)	95 (-44.51%)	46.11	159 (-0.62%)	160 (-6.54%)	43.17
graph_10	**183.6 (1.02)**	185	60	152 (-2.56%)	156 (-15.03%)	35.09	159 (0.0%)	159 (-13.4%)	33.84
graph_11	**198.0 (0.63)**	199	60	158 (-8.14%)	172 (-13.13%)	115.55	152 (-6.17%)	162 (-18.18%)	111.51
graph_12	**174.6 (0.49)**	175	60	152 (-1.3%)	154 (-11.8%)	27.93	155 (-1.9%)	158 (-9.51%)	26.94
graph_13	**169.4 (1.5)**	172	60	55 (-15.38%)	65 (-61.63%)	75.29	150 (-0.66%)	151 (-10.86%)	70.19
graph_14	**160.4 (2.06)**	163	60	149 (-6.29%)	159 (-0.87%)	13.14	156 (-1.27%)	158 (-1.5%)	12.73

Table 5.12: The results for $k = 50$. **DCCA-Same**: mean difference between TOPKLS is -20.79% (SD = 19.27%) with the mean difference between the end and the max score being -5.08% (SD = 6.3%). **DCCA-Mix**: mean difference between TOPKLS is -13.52% (SD = 7.15%) with the mean difference between the end and the max score being -2.36% (SD = 2.34%).

Results T-Test

The results of the dependent T-test between the percentage difference of *TOPKLS* with a cutoff time of 600 seconds and 60 seconds for both DCCA-Same and DCCA-Mix are as follow:

- If $k = 10$, then for DCCA-Same, the difference between a cutoff time of 60 seconds ($M = -21.68\%$, $SD = 8.13\%$) and 600 seconds ($M = -23.18\%$, $SD =$

7.85%) is significant, T(14)=11.1383, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = −23.57%, SD = 9.28%) or 600 seconds (M = −25.05%, SD = 8.82%) is significant, T(14)=9.9108, $p < 0.01$.

- If $k = 30$, then for DCCA-Same, the difference between a cutoff time of 60 seconds (M = −19.98%, SD = 10.78%) and 600 seconds (M = −20.93%, SD = 10.74%) is significant, T(14)=14.1680, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = −21.48%, SD = 16.27%) or 600 seconds (M = −22.43%, SD = 16.01%) is significant, T(14)=9.6207, $p < 0.01$.

- If $k = 50$, then for DCCA-Same, the difference between a cutoff time of 60 seconds (M = −20.79%, SD = 19.27%) and 600 seconds (M = −21.50%, SD = 19.06%) is significant, T(14)=7.6096, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = −13.52%, SD = 7.15%) or 600 seconds (M = −14.29%, SD = 6.90%) is significant, T(14)=8.5333, $p < 0.01$.

5.1.3 Real-world Graphs

The results of the real-world graphs in tables 5.13, 5.14 and 5.15 show that *TOPKLS* is the better option for all graphs. DCCA-Mix and DCCA-Same timeout on some graphs, because it took longer than 900 seconds. We see a significant difference in the results between those and the graphs they did not timeout[1], indicating that if they had finished on those graphs, that the results might have been better. If we only look at the graphs they both finished and compare them to each other; we see that DCCA-Same is better when $k = 10$ and $k = 30$, and DCCA-Mix is better when $k = 50$, which also saw in the previous two experiments (sections 5.1.1 and 5.1.2)

TOPKLS cutoff of 600 seconds

When we combine the results of all the three tested values of k (tables 5.13, 5.14 and 5.15), we see that for all the graphs, the mean difference between *TOPKLS* and DCCA-Same is −42.95% (SD = 26.4) and the mean difference between *TOPKLS* and DCCA-Mix is −42.32% (SD = 27.84%). The mean difference between the end and the max score for DCCA-Same is −18.1% (SD = 18.12%) and for DCCA-Mix is −16.83% (SD = 15.94%).

For the graphs that DCCA finished, the mean difference between *TOPKLS* and DCCA-Same is −31.69% (SD = 20.06) and the mean difference between *TOPKLS* and DCCA-Mix is −30.11% (SD = 22.03%). The mean difference between the end and the max score for DCCA-Same is −12.49% (SD = 14.29%) and for DCCA-Mix is −10.03% (SD = 10.89%).

Lastly, for the graphs that DCCA timed out at, the mean difference between *TOPKLS* and DCCA-Same is −65.49% (SD = 23.11) and the mean difference between *TOPKLS* and DCCA-Mix is −66.73% (SD = 21.98%). The mean difference between the end and the max score for DCCA-Same is −29.31% (SD = 20.13%) and for DCCA-Mix is −30.42% (SD = 15.3%).

[1]The graphs at which the runtime is less then 900 seconds

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
ca-GrQc	**246.0 (0.0)**	246	600	191 (-7.73%)	207 (-15.85%)	3.39	191 (-8.17%)	208 (-15.45%)	3.38
ca-netscience	**68.0 (0.0)**	68	600	65 (0.0%)	65 (-4.41%)	0.15	62 (0.0%)	62 (-8.82%)	0.15
ia-email-univ	**75.0 (0.0)**	75	600	34 (-12.82%)	39 (-48.0%)	2.57	34 (-15.0%)	40 (-46.67%)	2.66
ia-infect-dublin	**110.0 (0.0)**	110	600	43 (-32.81%)	64 (-41.82%)	0.95	42 (-22.22%)	54 (-50.91%)	1.01
inf-power	**46.0 (0.0)**	46	600	44 (-2.22%)	45 (-2.17%)	4.63	28 (-28.21%)	39 (-15.22%)	4.59
rt-retweet	**24.0 (0.0)**	24	600	21 (0.0%)	21 (-12.5%)	0.07	19 (0.0%)	19 (-20.83%)	0.07
sc-shipsec1	**232.8 (2.04)**	236	600	204 (-7.27%)	220 (-5.5%)	**900.0**	98 (-41.67%)	168 (-27.84%)	**900.0**
soc-buzznet	**152.6 (2.73)**	158	600	27 (-40.0%)	45 (-70.51%)	**900.0**	28 (-37.78%)	45 (-70.51%)	**900.0**
socfb-CMU	**314.6 (1.74)**	317	600	43 (-47.56%)	82 (-73.94%)	**900.0**	55 (-24.66%)	73 (-76.8%)	**900.0**
tech-RL-caida	**103.4 (1.02)**	105	600	26 (-50.94%)	53 (-48.74%)	**900.0**	25 (-46.81%)	47 (-54.55%)	**900.0**
tech-WHOIS	**280.6 (1.2)**	282	600	64 (-7.25%)	69 (-75.41%)	**900.0**	51 (-25.0%)	68 (-75.77%)	**900.0**
tech-internet-as	**59.2 (1.17)**	61	600	31 (-32.61%)	46 (-22.3%)	88.58	36 (-12.2%)	41 (-30.74%)	88.52
tech-routers-rf	**95.4 (0.49)**	96	600	27 (-34.15%)	41 (-57.02%)	2.83	26 (-36.59%)	41 (-57.02%)	2.78
web-arabic-2005	**738.0 (0.0)**	738	600	500 (-0.4%)	502 (-31.98%)	372.54	439 (-5.79%)	466 (-36.86%)	372.48
web-spam	**126.0 (0.0)**	126	600	44 (-33.33%)	66 (-47.62%)	53.91	44 (-15.38%)	52 (-58.73%)	53.73

Table 5.13: The results for $k = 10$. If the runtime for DCCA is stated in bold, it means that DCCA terminated early on those graphs, because it exceeded the 900 seconds timeout limit. **DCCA-Same**: mean difference between TOPKLS is -37.18% (SD = 25.79%) for all graphs, -28.37% (SD = 19.65%) for the finished graphs and -54.82% (SD = 29.6%) for the not finished graphs and with the mean difference between the end and the max score being -20.61% (SD = 18.59%) for all graphs, -15.61% (SD = 15.67%) for the finished graphs and -30.6% (SD = 21.68%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKLS is -43.11% (SD = 22.73%) for all graphs, -34.12% (SD = 18.59%) for the finished graphs and -61.09% (SD = 20.61%) for the not finished graphs with the mean difference between the end and the max score being -21.3% (SD = 14.93%) for all graphs, -14.36% (SD = 11.91%) for the finished graphs and -35.18% (SD = 9.98%) for the not finished graphs.

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
ca-GrQc	**422.4 (0.49)**	423	600	380 (-0.52%)	382 (-9.56%)	5.19	380 (-3.31%)	393 (-6.96%)	5.3
ca-netscience	**158.0 (0.0)**	158	600	128 (-2.29%)	131 (-17.09%)	0.21	144 (0.0%)	144 (-8.86%)	0.2
ia-email-univ	**173.2 (0.75)**	174	600	82 (-1.2%)	83 (-52.08%)	4.05	89 (-2.2%)	91 (-47.46%)	4.06
ia-infect-dublin	**215.2 (0.4)**	216	600	113 (-0.88%)	114 (-47.03%)	1.46	97 (-9.35%)	107 (-50.28%)	1.51
inf-power	**120.8 (0.75)**	122	600	82 (-14.58%)	96 (-20.53%)	7.01	115 (0.0%)	115 (-4.8%)	6.97
rt-retweet	**62.0 (0.0)**	62	600	33 (-23.26%)	43 (-30.65%)	0.08	51 (-3.77%)	53 (-14.52%)	0.08
sc-shipsec1	**627.8 (2.04)**	630	600	337 (-21.99%)	432 (-31.19%)	**900.0**	479 (-3.43%)	496 (-20.99%)	**900.0**
soc-buzznet	**333.6 (3.88)**	338	600	39 (-27.78%)	54 (-83.81%)	**900.0**	37 (-27.45%)	51 (-84.71%)	**900.0**
socfb-CMU	**685.4 (2.06)**	687	600	85 (-39.29%)	140 (-79.57%)	**900.0**	73 (-38.66%)	119 (-82.64%)	**900.0**
tech-RL-caida	**242.0 (2.28)**	246	600	62 (-17.33%)	75 (-69.01%)	**900.0**	65 (-26.14%)	88 (-63.64%)	**900.0**
tech-WHOIS	**486.6 (1.5)**	489	600	29 (-69.79%)	96 (-80.27%)	**900.0**	33 (-65.62%)	96 (-80.27%)	**900.0**
tech-internet-as	**123.4 (1.02)**	125	600	88 (-10.2%)	98 (-20.58%)	134.7	88 (-17.76%)	107 (-13.29%)	132.23
tech-routers-rf	**180.4 (0.49)**	181	600	68 (-19.05%)	84 (-53.44%)	4.26	67 (-20.24%)	84 (-53.44%)	4.25
web-arabic-2005	**1769.0 (0.0)**	1769	600	1294 (-8.87%)	1420 (-19.73%)	497.4	1709 (-2.84%)	1759 (-0.57%)	500.35
web-spam	**267.6 (1.5)**	270	600	73 (-28.43%)	102 (-61.88%)	82.16	81 (-16.49%)	97 (-63.75%)	82.18

Table 5.14: The results for $k = 30$. If the runtime for DCCA is stated in bold, it means that DCCA terminated early on those graphs, because it exceeded the 900 seconds timeout limit. **DCCA-Same**: mean difference between TOPKLS is -45.09% (SD = 25.63%) for all graphs, -33.26% (SD = 18.58%) for the finished graphs and -68.77% (SD = 21.72%) for the not finished graphs and with the mean difference between the end and the max score being -19.03% (SD = 18.23%) for all graphs, -10.93% (SD = 10.11%) for the finished graphs and -35.24% (SD = 20.99%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKLS is -39.74% (SD = 31.03%) for all graphs, -26.39% (SD = 24.2%) for the finished graphs and -66.45% (SD = 26.74%) for the not finished graphs with the mean difference between the end and the max score being -15.82% (SD = 18.16%) for all graphs, -7.6% (SD = 7.78%) for the finished graphs and -32.26% (SD = 22.62%) for the not finished graphs.

	TOPKLS			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
ca-GrQc	**546.6 (0.49)**	547	600	507 (-3.24%)	524 (-4.13%)	6.83	500 (-3.29%)	517 (-5.42%)	7.03
ca-netscience	**222.0 (0.0)**	222	600	185 (0.0%)	185 (-16.67%)	0.25	207 (0.0%)	207 (-6.76%)	0.24
ia-email-univ	**252.8 (0.75)**	254	600	121 (0.0%)	121 (-52.14%)	5.3	129 (0.0%)	129 (-48.97%)	5.2
ia-infect-dublin	**280.6 (0.49)**	281	600	122 (-1.61%)	124 (-55.81%)	1.93	123 (-3.91%)	128 (-54.38%)	1.97
inf-power	**180.8 (0.75)**	182	600	146 (-1.35%)	148 (-18.14%)	9.25	146 (-8.75%)	160 (-11.5%)	9.31
rt-retweet	**82.0 (0.0)**	82	600	55 (-14.06%)	64 (-21.95%)	0.08	73 (-1.35%)	74 (-9.76%)	0.08
sc-shipsec1	**997.8 (2.71)**	1000	600	549 (-5.02%)	578 (-42.07%)	**900.0**	554 (-4.81%)	582 (-41.67%)	**900.0**
soc-buzznet	**483.6 (3.83)**	489	600	39 (-36.07%)	61 (-87.39%)	**900.0**	39 (-30.36%)	56 (-88.42%)	**900.0**
socfb-CMU	**955.6 (4.13)**	961	600	159 (-23.92%)	209 (-78.13%)	**900.0**	173 (-30.8%)	250 (-73.84%)	**900.0**
tech-RL-caida	**362.4 (2.06)**	366	600	93 (-3.12%)	96 (-73.51%)	**900.0**	81 (-15.62%)	96 (-73.51%)	**900.0**
tech-WHOIS	**620.0 (3.58)**	625	600	60 (-42.31%)	104 (-83.23%)	**900.0**	55 (-37.5%)	88 (-85.81%)	**900.0**
tech-internet-as	**183.0 (1.1)**	185	600	48 (-52.48%)	101 (-44.81%)	185.85	68 (-39.29%)	112 (-38.8%)	186.23
tech-routers-rf	**246.8 (1.33)**	249	600	89 (-28.8%)	125 (-49.35%)	5.62	91 (-22.88%)	118 (-52.19%)	5.81
web-arabic-2005	**2789.0 (0.0)**	2789	600	2488 (-3.49%)	2578 (-7.57%)	690.79	2550 (0.0%)	2550 (-8.57%)	697.03
web-spam	**382.0 (2.1)**	385	600	132 (-4.35%)	138 (-63.87%)	110.07	143 (-2.05%)	146 (-61.78%)	111.68

Table 5.15: The results for $k = 50$. If the runtime for DCCA is stated in bold, it means that DCCA terminated early on those graphs, because it exceeded the 900 seconds timeout limit. **DCCA-Same**: mean difference between TOPKLS is -46.58% (SD = 27.79%) for all graphs, -33.44% (SD = 21.95%) for the finished graphs and -72.86% (SD = 17.99%) for the not finished graphs and with the mean difference between the end and the max score being -14.65% (SD = 17.53%) for all graphs, -10.94% (SD = 17.1%) for the finished graphs and -22.09% (SD = 17.74%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKLS is -44.09% (SD = 29.75%) for all graphs, -29.81% (SD = 23.31%) for the finished graphs and -72.65% (SD = 18.6%) for the not finished graphs with the mean difference between the end and the max score being -13.37% (SD = 14.73%) for all graphs, -8.15% (SD = 12.96%) for the finished graphs and -23.82% (SD = 13.3%) for the not finished graphs.

TOPKLS cutoff of 60 seconds

When we combine the results of all the three tested values of k (tables 5.16, 5.17, and 5.18) for a cutoff of 60 seconds, we see that for all the graphs, the mean difference between *TOPKLS* and DCCA-Same is -42.45% (SD = 26.4%) and the mean difference between *TOPKLS* and DCCA-Mix is -41.81% (SD = 27.83%).

For the graphs that DCCA finished, the mean difference between *TOPKLS* and DCCA-Same is -31.2% (SD = 19.99%) and the mean difference between *TOPKLS* and DCCA-Mix is -29.62% (SD = 21.96%).

Lastly, for the graphs that DCCA timed out at, because it took longer then 900 seconds, the mean difference between *TOPKLS* and DCCA-Same is -64.94% (SD = 23.21%) and the mean difference between *TOPKLS* and DCCA-Mix is -66.21% (SD = 22.1%).

Graphs	TOPKLS			DCCA-Same			DCCA-Mix		
	Result	Max	Run Time	End Score	Max Score	Run Time	End Score	Max Score	Run Time
ca-GrQc	**246.0 (0.0)**	246	60	191 (-7.73%)	207 (-15.85%)	3.39	191 (-8.17%)	208 (-15.45%)	3.38
ca-netscience	**68.0 (0.0)**	68	60	65 (0.0%)	65 (-4.41%)	0.15	62 (0.0%)	62 (-8.82%)	0.15
ia-email-univ	**74.6 (0.49)**	75	60	34 (-12.82%)	39 (-47.72%)	2.57	34 (-15.0%)	40 (-46.38%)	2.66
ia-infect-dublin	**110.0 (0.0)**	110	60	43 (-32.81%)	64 (-41.82%)	0.95	42 (-22.22%)	54 (-50.91%)	1.01
inf-power	**46.0 (0.0)**	46	60	44 (-2.22%)	45 (-2.17%)	4.63	28 (-28.21%)	39 (-15.22%)	4.59
rt-retweet	**24.0 (0.0)**	24	60	21 (0.0%)	21 (-12.5%)	0.07	19 (0.0%)	19 (-20.83%)	0.07
sc-shipsec1	**230.6 (2.06)**	234	60	204 (-7.27%)	220 (-4.6%)	**900.0**	98 (-41.67%)	168 (-27.15%)	**900.0**
soc-buzznet	**147.6 (2.8)**	151	60	27 (-40.0%)	45 (-69.51%)	**900.0**	28 (-37.78%)	45 (-69.51%)	**900.0**
socfb-CMU	**310.2 (3.92)**	314	60	43 (-47.56%)	82 (-73.57%)	**900.0**	55 (-24.66%)	73 (-76.47%)	**900.0**
tech-RL-caida	**99.6 (1.85)**	102	60	26 (-50.94%)	53 (-46.79%)	**900.0**	25 (-46.81%)	47 (-52.81%)	**900.0**
tech-WHOIS	**276.8 (1.17)**	278	60	64 (-7.25%)	69 (-75.07%)	**900.0**	51 (-25.0%)	68 (-75.43%)	**900.0**
tech-internet-as	**56.6 (1.5)**	58	60	31 (-32.61%)	46 (-18.73%)	88.58	36 (-12.2%)	41 (-27.56%)	88.52
tech-routers-rf	**93.6 (0.8)**	95	60	27 (-34.15%)	41 (-56.2%)	2.83	26 (-36.59%)	41 (-56.2%)	2.78
web-arabic-2005	**737.8 (0.4)**	738	60	500 (-0.4%)	502 (-31.96%)	372.54	439 (-5.79%)	466 (-36.84%)	372.48
web-spam	**124.4 (0.49)**	125	60	44 (-33.33%)	66 (-46.95%)	53.91	44 (-15.38%)	52 (-58.2%)	53.73

Table 5.16: The results for $k = 10$. **DCCA-Same**: mean difference between TOPKLS is -36.52% (SD = 25.74%) for all graphs, -27.83% (SD = 19.57%) for the finished graphs and -53.91% (SD = 29.83%) for the not finished graphs and with the mean difference between the end and the max score being -20.61% (SD = 18.59%) for all graphs, -15.61% (SD = 15.67%) for the finished graphs and -30.6% (SD = 21.68%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKLS is -42.52% (SD = 22.62%) for all graphs, -33.64% (SD = 18.47%) for the finished graphs and -60.27% (SD = 20.8%) for the not finished graphs with the mean difference between the end and the max score being -21.3% (SD = 14.93%) for all graphs, -14.36% (SD = 11.91%) for the finished graphs and -35.18% (SD = 9.98%) for the not finished graphs.

Graphs	TOPKLS			DCCA-Same			DCCA-Mix		
	Result	Max	Run Time	End Score	Max Score	Run Time	End Score	Max Score	Run Time
ca-GrQc	**422.0 (0.0)**	422	60	380 (-0.52%)	382 (-9.48%)	5.19	380 (-3.31%)	393 (-6.87%)	5.3
ca-netscience	**158.0 (0.0)**	158	60	128 (-2.29%)	131 (-17.09%)	0.21	144 (0.0%)	144 (-8.86%)	0.2
ia-email-univ	**171.2 (0.98)**	173	60	82 (-1.2%)	83 (-51.52%)	4.05	89 (-2.2%)	91 (-46.85%)	4.06
ia-infect-dublin	**214.0 (0.63)**	215	60	113 (-0.88%)	114 (-46.73%)	1.46	97 (-9.35%)	107 (-50.0%)	1.51
inf-power	**119.4 (0.49)**	120	60	82 (-14.58%)	96 (-19.6%)	7.01	115 (0.0%)	115 (-3.69%)	6.97
rt-retweet	**62.0 (0.0)**	62	60	33 (-23.26%)	43 (-30.65%)	0.08	51 (-3.77%)	53 (-14.52%)	0.08
sc-shipsec1	**624.4 (3.2)**	630	60	337 (-21.99%)	432 (-30.81%)	**900.0**	479 (-3.43%)	496 (-20.56%)	**900.0**
soc-buzznet	**327.4 (5.99)**	338	60	39 (-27.78%)	54 (-83.51%)	**900.0**	37 (-27.45%)	51 (-84.42%)	**900.0**
socfb-CMU	**679.0 (4.15)**	687	60	85 (-39.29%)	140 (-79.38%)	**900.0**	73 (-38.66%)	119 (-82.47%)	**900.0**
tech-RL-caida	**233.8 (1.94)**	236	60	62 (-17.33%)	75 (-67.92%)	**900.0**	65 (-26.14%)	88 (-62.36%)	**900.0**
tech-WHOIS	**483.8 (2.04)**	487	60	29 (-69.79%)	96 (-80.16%)	**900.0**	33 (-65.62%)	96 (-80.16%)	**900.0**
tech-internet-as	**120.2 (2.48)**	123	60	88 (-10.2%)	98 (-18.47%)	134.7	88 (-17.76%)	107 (-10.98%)	132.23
tech-routers-rf	**178.4 (0.49)**	179	60	68 (-19.05%)	84 (-52.91%)	4.26	67 (-20.24%)	84 (-52.91%)	4.25
web-arabic-2005	**1768.6 (0.49)**	1769	60	1294 (-8.87%)	1420 (-19.71%)	497.4	1709 (-2.84%)	1759 (-0.54%)	500.35
web-spam	**263.8 (0.98)**	265	60	73 (-28.43%)	102 (-61.33%)	82.16	81 (-16.49%)	97 (-63.23%)	82.18

Table 5.17: The results for $k = 30$. **DCCA-Same**: mean difference between TOPKLS is -44.62% (SD = 25.68%) for all graphs, -32.75% (SD = 18.59%) for the finished graphs and -68.36% (SD = 21.79%) for the not finished graphs and with the mean difference between the end and the max score being -19.03% (SD = 18.23%) for all graphs, -10.93% (SD = 10.11%) for the finished graphs and -35.24% (SD = 20.99%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKLS is -39.23% (SD = 31.11%) for all graphs, -25.84% (SD = 24.23%) for the finished graphs and -66.0% (SD = 26.87%) for the not finished graphs with the mean difference between the end and the max score being -15.82% (SD = 18.16%) for all graphs, -7.6% (SD = 7.78%) for the finished graphs and -32.26% (SD = 22.62%) for the not finished graphs.

Graphs	TOPKLS Result	TOPKLS Max	TOPKLS Run Time	DCCA-Same End Score	DCCA-Same Max Score	DCCA-Same Run Time	DCCA-Mix End Score	DCCA-Mix Max Score	DCCA-Mix Run Time
ca-GrQc	**544.6 (1.36)**	546	60	507 (-3.24%)	524 (-3.78%)	6.83	500 (-3.29%)	517 (-5.07%)	7.03
ca-netscience	**221.2 (0.4)**	222	60	185 (0.0%)	185 (-16.37%)	0.25	207 (0.0%)	207 (-6.42%)	0.24
ia-email-univ	**250.8 (1.33)**	253	60	121 (0.0%)	121 (-51.75%)	5.3	129 (0.0%)	129 (-48.56%)	5.2
ia-infect-dublin	**279.2 (0.4)**	280	60	122 (-1.61%)	124 (-55.59%)	1.93	123 (-3.91%)	128 (-54.15%)	1.97
inf-power	**179.4 (0.49)**	180	60	146 (-1.35%)	148 (-17.5%)	9.25	146 (-8.75%)	160 (-10.81%)	9.31
rt-retweet	**82.0 (0.0)**	82	60	55 (-14.06%)	64 (-21.95%)	0.08	73 (-1.35%)	74 (-9.76%)	0.08
sc-shipsec1	**993.6 (3.2)**	999	60	549 (-5.02%)	578 (-41.83%)	**900.0**	554 (-4.81%)	582 (-41.43%)	**900.0**
soc-buzznet	**477.4 (6.59)**	489	60	39 (-36.07%)	61 (-87.22%)	**900.0**	39 (-30.36%)	56 (-88.27%)	**900.0**
socfb-CMU	**943.4 (7.5)**	958	60	159 (-23.92%)	209 (-77.85%)	**900.0**	173 (-30.8%)	250 (-73.5%)	**900.0**
tech-RL-caida	**353.8 (1.94)**	356	60	93 (-3.12%)	96 (-72.87%)	**900.0**	81 (-15.62%)	96 (-72.87%)	**900.0**
tech-WHOIS	**614.4 (3.01)**	620	60	60 (-42.31%)	104 (-83.07%)	**900.0**	55 (-37.5%)	88 (-85.68%)	**900.0**
tech-internet-as	**179.2 (1.94)**	182	60	48 (-52.48%)	101 (-43.64%)	185.85	68 (-39.29%)	112 (-37.5%)	186.23
tech-routers-rf	**243.0 (0.63)**	244	60	89 (-28.8%)	125 (-48.56%)	5.62	91 (-22.88%)	118 (-51.44%)	5.81
web-arabic-2005	**2788.6 (0.49)**	2789	60	2488 (-3.49%)	2578 (-7.55%)	690.79	2550 (0.0%)	2550 (-8.56%)	697.03
web-spam	**378.2 (1.6)**	381	60	132 (-4.35%)	138 (-63.51%)	110.07	143 (-2.05%)	146 (-61.4%)	111.68

Table 5.18: The results for $k = 50$. **DCCA-Same**: mean difference between TOPKLS is -46.2% (SD = 27.77%) for all graphs, -33.02% (SD = 21.83%) for the finished graphs and -72.57% (SD = 18.01%) for the not finished graphs and with the mean difference between the end and the max score being -14.65% (SD = 17.53%) for all graphs, -10.94% (SD = 17.1%) for the finished graphs and -22.09% (SD = 17.74%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKLS is -43.69% (SD = 29.75%) for all graphs, -29.37% (SD = 23.19%) for the finished graphs and -72.35% (SD = 18.63%) for the not finished graphs with the mean difference between the end and the max score being -13.37% (SD = 14.73%) for all graphs, -8.15% (SD = 12.96%) for the finished graphs and -23.82% (SD = 13.3%) for the not finished graphs.

Results T-Test

The results of the dependent T-test between the percentage difference of *TOPKLS* with a cutoff time of 600 seconds and 60 seconds for both DCCA-Same and DCCA-Mix are as follow:

- If $k = 10$, then for DCCA-Same, the difference between a cutoff time of 60 seconds (M = −36.52%, SD = 25.74%) and 600 seconds (M = −37.18%, SD = 25.79%) is insignificant, T(14)=2.6338, p=0.0196. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = −42.52, SD = 22.62%) or 600 seconds (M = −43.11%, SD = 22.73%) is insignificant, T(14)=2.6539, p=0.0189.

- If $k = 30$, then for DCCA-Same, the difference between a cutoff time of 60 seconds (M = −44.62%, SD = 25.68%) and 600 seconds (M = −45.09%, SD = 25.63%) is significant, T(14)=3.3031, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = −39.23%, SD = 31.11%) or 600 seconds (M = −39.74%, SD = 31.03%) is significant, T(14)=3.1850, $p < 0.01$.

- If $k = 50$, then for DCCA-Same, the difference between a cutoff time of 60 seconds (M = −46.20%, SD = 27.77%) and 600 seconds (M = −46.58%, SD = 27.79%) is significant, T(14)=4.7262, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = −43.69%, SD = 29.75%) or 600 seconds (M = −44.09%, SD = 29.75%) is significant, T(14)=4.5827, $p < 0.01$.

5.2 Diversified Top-k Weighted Clique Search

The results of the experiments of the diversified top-k weighted clique search problem (DTKWC) show that the baseline, *TOPKWCLQ*, is significantly better for both cutoff times. We show this by stating the average percentage between the max score of DCCA and the mean results of *TOPKWCLQ*. For a cutoff of 600 seconds, this average difference is for DCCA-Same -35.5% (SD=13.53%) and DCCA-Mix -37.49% (SD=12.39%). This difference becomes smaller when the cutoff time is 60 seconds; however, it is still -34.83% (SD=13.57%) for DCCA-Same and -36.84% (SD=12.48%) for DCCA-Mix.

The difference between *TOPKWCLQ* and DCCA-Same and DCCA-Mix is larger than it was between *TOPKLS* and DCCA for the diversified top-k clique search problem (DTKC) (section 5.1). This difference indicates that there is a reason why DCCA could find better results for DTKC than it could for DTKWC. Our discussion of the results will explain why this happened in section 6.1.

5.2.1 Dual Barabási–Albert model - Same Parameters

The results for DTKWC in tables 5.19, 5.20, and 5.21 show that *TOPKWCLQ*, with a cutoff time 600 seconds is significantly better than DCCA, especially compared to the same graphs for the diversified top-k clique search problem in section 5.1.1. Lowering the cutoff time of *TOPKWCLQ* does not change this (tables 5.22, 5.23, and 5.24). Nevertheless, when we compare DCCA-Mix and DCCA-Same, we see similar behaviour that DCCA-Same is better at $k = 10$ and $k = 30$, and DCCA-Mix when $k = 50$.

TOPKWCLQ cutoff of 600 seconds

When we combine the results of all the three tested values of k (tables 5.19, 5.20, and 5.21), we see that the mean difference between *TOPKWCLQ* and DCCA-Same is -24.24% (SD = 3.01) and the mean difference between *TOPKWCLQ* and DCCA-Mix is -29.5% (SD = 4.49%). The mean difference between the end and the max score for DCCA-Same is -5.66% (SD = 5.45%) and for DCCA-Mix is -10.68% (SD = 4.66%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**346.6 (3.2)**	350	600	264 (-4.69%)	277 (-20.08%)	7.04	153 (-31.39%)	223 (-35.66%)	6.79
graph_1	**331.6 (1.96)**	334	600	213 (-19.62%)	265 (-20.08%)	6.99	137 (-34.13%)	208 (-37.27%)	6.81
graph_2	**344.6 (2.15)**	348	600	262 (0.0%)	262 (-23.97%)	6.41	132 (-33.67%)	199 (-42.25%)	6.28
graph_3	**339.0 (3.16)**	345	600	246 (-5.38%)	260 (-23.3%)	6.43	154 (-13.97%)	179 (-47.2%)	6.26
graph_4	**333.6 (1.96)**	337	600	226 (0.0%)	226 (-32.25%)	6.78	120 (-21.05%)	152 (-54.44%)	6.5
graph_5	**368.8 (3.31)**	373	600	279 (-3.12%)	288 (-21.91%)	6.72	166 (-31.69%)	243 (-34.11%)	6.54
graph_6	**327.4 (1.74)**	329	600	201 (-12.99%)	231 (-29.44%)	6.54	119 (-35.68%)	185 (-43.49%)	6.37
graph_7	**350.6 (1.85)**	353	600	253 (-4.53%)	265 (-24.42%)	6.64	155 (-27.91%)	215 (-38.68%)	6.29
graph_8	**343.8 (1.6)**	346	600	251 (0.0%)	251 (-26.99%)	6.33	174 (-16.35%)	208 (-39.5%)	6.05
graph_9	**388.8 (1.6)**	391	600	231 (-27.13%)	317 (-18.47%)	7.43	202 (-17.55%)	245 (-36.99%)	7.27

Table 5.19: The results for $k = 10$. **DCCA-Same**: mean difference between TOPKWCLQ is -24.09% (SD = 4.38%) with the mean difference between the end and the max score being -7.75% (SD = 9.25%). **DCCA-Mix**: mean difference between TOPKWCLQ is -40.96% (SD = 6.15%) with the mean difference between the end and the max score being -26.34% (SD = 8.28%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**759.6 (4.76)**	764	600	567 (0.0%)	567 (-25.36%)	10.6	**564 (-0.53%)**	567 (-25.36%)	10.92
graph_1	**763.2 (2.99)**	768	600	525 (-7.41%)	567 (-25.71%)	10.61	526 (-1.87%)	536 (-29.77%)	10.74
graph_2	**756.4 (4.5)**	761	600	511 (-12.8%)	586 (-22.53%)	9.81	544 (-1.27%)	551 (-27.15%)	10.01
graph_3	**737.4 (1.85)**	739	600	497 (-15.33%)	587 (-20.4%)	9.63	498 (-9.62%)	551 (-25.28%)	9.64
graph_4	**743.2 (2.48)**	746	600	516 (-4.27%)	539 (-27.48%)	10.17	524 (-4.73%)	550 (-26.0%)	10.48
graph_5	**772.8 (3.12)**	777	600	584 (-2.83%)	601 (-22.23%)	10.18	625 (-6.58%)	669 (-13.43%)	10.42
graph_6	**738.6 (3.61)**	745	600	516 (-4.27%)	539 (-27.02%)	10.01	539 (0.0%)	539 (-27.02%)	10.28
graph_7	**776.6 (2.15)**	780	600	541 (-10.13%)	602 (-22.48%)	9.92	588 (-4.23%)	614 (-20.94%)	10.03
graph_8	**750.6 (4.59)**	757	600	566 (-6.45%)	605 (-19.4%)	9.48	566 (-3.25%)	585 (-22.06%)	9.68
graph_9	**794.4 (6.22)**	803	600	548 (-6.0%)	583 (-26.61%)	11.25	549 (-4.52%)	575 (-27.62%)	11.29

Table 5.20: The results for $k = 30$. **DCCA-Same**: mean difference between TOPKWCLQ is -23.92% (SD = 2.88%) with the mean difference between the end and the max score being -6.95% (SD = 4.66%). **DCCA-Mix**: mean difference between TOPKWCLQ is -24.46% (SD = 4.66%) with the mean difference between the end and the max score being -3.66% (SD = 2.95%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**1131.6 (6.56)**	1143	600	861 (0.0%)	861 (-23.91%)	14.08	879 (-0.45%)	883 (-21.97%)	14.64
graph_1	**1147.2 (3.66)**	1153	600	804 (-7.05%)	865 (-24.6%)	14.33	850 (0.0%)	850 (-25.91%)	14.33
graph_2	**1130.6 (3.72)**	1134	600	822 (-4.97%)	865 (-23.49%)	13.05	807 (-4.95%)	849 (-24.91%)	12.97
graph_3	**1106.0 (2.37)**	1109	600	801 (-1.35%)	812 (-26.58%)	12.9	827 (-7.7%)	896 (-18.99%)	14.05
graph_4	**1115.6 (1.74)**	1118	600	843 (-0.82%)	850 (-23.81%)	13.55	823 (0.0%)	823 (-26.23%)	13.8
graph_5	**1144.2 (2.48)**	1147	600	836 (-0.36%)	839 (-26.67%)	13.87	878 (0.0%)	878 (-23.27%)	13.27
graph_6	**1112.6 (5.12)**	1120	600	838 (-0.24%)	840 (-24.5%)	13.23	839 (-1.06%)	848 (-23.78%)	14.11
graph_7	**1156.0 (3.52)**	1162	600	892 (-1.55%)	906 (-21.63%)	12.91	922 (-1.39%)	935 (-19.12%)	13.27
graph_8	**1123.6 (4.08)**	1131	600	835 (-1.65%)	849 (-24.44%)	12.77	883 (0.0%)	883 (-21.41%)	12.53
graph_9	**1166.6 (6.28)**	1175	600	805 (-4.73%)	845 (-27.57%)	15.1	830 (-4.82%)	872 (-25.25%)	15.66

Table 5.21: The results for $k = 50$. **DCCA-Same**: mean difference between TOPKWCLQ is -24.72% (SD = 1.77%) with the mean difference between the end and the max score being -2.27% (SD = 2.43%). **DCCA-Mix**: mean difference between TOPKWCLQ is -23.08% (SD = 2.64%) with the mean difference between the end and the max score being -2.04% (SD = 2.76%).

TOPKWCLQ cutoff of 60 seconds

When we combine the results of all the three tested values of k (tables 5.22, 5.23, and 5.24) with a cutoff of 60 seconds, we see that the mean difference between *TOPKWCLQ* and DCCA-Same is -23.42% (SD = 3.09) and the mean difference between *TOPKWCLQ* and DCCA-Mix is -28.76% (SD = 4.58%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**340.2 (1.94)**	342	60	264 (-4.69%)	277 (-18.58%)	7.04	153 (-31.39%)	223 (-34.45%)	6.79
graph_1	**326.8 (2.14)**	330	60	213 (-19.62%)	265 (-18.91%)	6.99	137 (-34.13%)	208 (-36.35%)	6.81
graph_2	**339.2 (1.94)**	342	60	262 (0.0%)	262 (-22.76%)	6.41	132 (-33.67%)	199 (-41.33%)	6.28
graph_3	**333.0 (4.73)**	338	60	246 (-5.38%)	260 (-21.92%)	6.43	154 (-13.97%)	179 (-46.25%)	6.26
graph_4	**331.6 (2.8)**	337	60	226 (0.0%)	226 (-31.85%)	6.78	120 (-21.05%)	152 (-54.16%)	6.5
graph_5	**364.0 (0.63)**	365	60	279 (-3.12%)	288 (-20.88%)	6.72	166 (-31.69%)	243 (-33.24%)	6.54
graph_6	**324.2 (2.04)**	328	60	201 (-12.99%)	231 (-28.75%)	6.54	119 (-35.68%)	185 (-42.94%)	6.37
graph_7	**347.4 (1.5)**	349	60	253 (-4.53%)	265 (-23.72%)	6.64	155 (-27.91%)	215 (-38.11%)	6.29
graph_8	**334.4 (1.2)**	336	60	251 (0.0%)	251 (-24.94%)	6.33	174 (-16.35%)	208 (-37.8%)	6.05
graph_9	**381.8 (1.94)**	385	60	231 (-27.13%)	317 (-16.97%)	7.43	202 (-17.55%)	245 (-35.83%)	7.27

Table 5.22: The results for $k = 10$. **DCCA-Same**: mean difference between TOPKQ-CLQ is -22.93% (SD = 4.64%) with the mean difference between the end and the max score being -7.75% (SD = 9.25%). **DCCA-Mix**: mean difference between TOPKQ-CLQ is -40.05% (SD = 6.37%) with the mean difference between the end and the max score being -26.34% (SD = 8.28%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**749.4 (7.09)**	763	60	567 (0.0%)	567 (-24.34%)	10.6	564 (-0.53%)	567 (-24.34%)	10.92
graph_1	**755.0 (3.74)**	759	60	525 (-7.41%)	567 (-24.9%)	10.61	526 (-1.87%)	536 (-29.01%)	10.74
graph_2	**750.0 (6.48)**	759	60	511 (-12.8%)	586 (-21.87%)	9.81	544 (-1.27%)	551 (-26.53%)	10.01
graph_3	**730.4 (1.62)**	733	60	497 (-15.33%)	587 (-19.63%)	9.63	498 (-9.62%)	551 (-24.56%)	9.64
graph_4	**740.6 (2.87)**	745	60	516 (-4.27%)	539 (-27.22%)	10.17	524 (-4.73%)	550 (-25.74%)	10.48
graph_5	**765.2 (6.37)**	776	60	584 (-2.83%)	601 (-21.46%)	10.18	625 (-6.58%)	669 (-12.57%)	10.42
graph_6	**731.2 (1.17)**	733	60	516 (-4.27%)	539 (-26.29%)	10.01	539 (0.0%)	539 (-26.29%)	10.28
graph_7	**770.0 (3.85)**	775	60	541 (-10.13%)	602 (-21.82%)	9.92	588 (-4.23%)	614 (-20.26%)	10.03
graph_8	**745.2 (6.05)**	757	60	566 (-6.45%)	605 (-18.81%)	9.48	566 (-3.25%)	585 (-21.5%)	9.68
graph_9	**781.8 (4.4)**	790	60	548 (-6.0%)	583 (-25.43%)	11.25	549 (-4.52%)	575 (-26.45%)	11.29

Table 5.23: The results for $k = 30$. **DCCA-Same**: mean difference between TOPKQ-CLQ is -23.18% (SD = 2.86%) with the mean difference between the end and the max score being -6.95% (SD = 4.66%). **DCCA-Mix**: mean difference between TOPKQ-CLQ is -23.72% (SD = 4.67%) with the mean difference between the end and the max score being -3.66% (SD = 2.95%).

Graphs	TOPKWCLQ Result	Max	Run Time	DCCA-Same End Score	DCCA-Same Max Score	Run Time	DCCA-Mix End Score	DCCA-Mix Max Score	Run Time
graph_0	**1123.0 (6.36)**	1135	60	861 (0.0%)	861 (-23.33%)	14.08	879 (-0.45%)	883 (-21.37%)	14.64
graph_1	**1137.6 (5.78)**	1143	60	804 (-7.05%)	865 (-23.96%)	14.33	850 (0.0%)	850 (-25.28%)	14.33
graph_2	**1122.8 (6.05)**	1133	60	822 (-4.97%)	865 (-22.96%)	13.05	807 (-4.95%)	849 (-24.39%)	12.97
graph_3	**1099.0 (5.55)**	1109	60	801 (-1.35%)	812 (-26.11%)	12.9	827 (-7.7%)	896 (-18.47%)	14.05
graph_4	**1113.2 (1.94)**	1115	60	843 (-0.82%)	850 (-23.64%)	13.55	823 (0.0%)	823 (-26.07%)	13.8
graph_5	**1137.2 (4.26)**	1144	60	836 (-0.36%)	839 (-26.22%)	13.87	878 (0.0%)	878 (-22.79%)	13.27
graph_6	**1102.6 (4.63)**	1109	60	838 (-0.24%)	840 (-23.82%)	13.23	839 (-1.06%)	848 (-23.09%)	14.11
graph_7	**1144.8 (4.45)**	1152	60	892 (-1.55%)	906 (-20.86%)	12.91	922 (-1.39%)	935 (-18.33%)	13.27
graph_8	**1118.0 (6.87)**	1131	60	835 (-1.65%)	849 (-24.06%)	12.77	883 (0.0%)	883 (-21.02%)	12.53
graph_9	**1153.0 (3.74)**	1157	60	805 (-4.73%)	845 (-26.71%)	15.1	830 (-4.82%)	872 (-24.37%)	15.66

Table 5.24: The results for $k = 50$. **DCCA-Same**: mean difference between TOPKQ-CLQ is -24.17% (SD = 1.77%) with the mean difference between the end and the max score being -2.27% (SD = 2.43%). **DCCA-Mix**: mean difference between TOPKQ-CLQ is -22.52% (SD = 2.69%) with the mean difference between the end and the max score being -2.04% (SD = 2.76%).

Results T-Test

The results of the dependent T-test between the percentage difference of *TOPKWCLQ* with a cutoff time of 600 seconds and 60 seconds for both DCCA-Same and DCCA-Mix are as follow:

- If $k = 10$, then for DCCA-Same, the difference between a cutoff time of 60 seconds ($M = -22.93\%$, $SD = 4.64\%$) and 600 seconds ($M = -24.09\%$, $SD = 4.38\%$) is significant, $T(9)=7.6443$, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds ($M = -40.05\%$, $SD = 6.37\%$) or 600 seconds ($M = -40.96\%$, $SD = 6.15\%$) is significant, $T(9)=7.2496$, $p < 0.01$.
- If $k = 30$, then for DCCA-Same, the difference between a cutoff time of 60 seconds ($M = -23.18\%$, $SD = 2.86\%$) and 600 seconds ($M = -23.92\%$, $SD = 2.88\%$) is significant, $T(9)=9.5226$, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds ($M = -23.72\%$, $SD = 4.67\%$) or 600 seconds ($M = -24.46\%$, $SD = 4.66\%$) is significant, $T(9)=9.4219$, $p < 0.01$.
- If $k = 50$, then for DCCA-Same, the difference between a cutoff time of 60 seconds ($M = -24.17\%$, $SD = 1.77\%$) and 600 seconds ($M = -24.72\%$, $SD = 1.77\%$) is significant, $T(9)=8.7088$, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds ($M = -22.52\%$, $SD = 2.69\%$) or 600 seconds ($M = -23.08\%$, $SD = 2.64\%$) is significant, $T(9)=8.7045$, $p < 0.01$.

5.2.2 Dual Barabási–Albert model - Random Parameters

The results for DTKWC in tables 5.25, 5.26, and 5.27 show that *TOPKWCLQ*, with a cutoff time of 600 seconds, is significantly better than DCCA, with a more significant difference than the results in section 5.2.1. The difference between the two graph sets we also saw in section 5.1, indicating that similar reasons that explain the difference between the two graph sets, in both problems. Again, lowering the cutoff time to 60

seconds does not impact the results (tables 5.28, 5.29, and 5.30). Interesting to note, is that again DCCA-Same is better at $k = 10$ and $k = 30$ and DCCA-Mix is better at $k = 50$, when the two compared to each other. This repeated behaviour indicates that this is no coincidence and that there is a reason why this must occur at every graph set, so far. We will explain this behaviour for this in section 6.1.

TOPKWCLQ cutoff of 600 seconds

When we combine the results of all the three tested values of k (tables 5.25, 5.26, and 5.27), we see that the mean difference between *TOPKWCLQ* and DCCA-Same is -34.9% (SD = 11.37) and the mean difference between *TOPKWCLQ* and DCCA-Mix is -34.64% (SD = 7.65%). The mean difference between the end and the max score for DCCA-Same is -12.19% (SD = 10.63%) and for DCCA-Mix is -15.56% (SD = 7.38%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**437.0 (4.69)**	444	600	286 (-15.13%)	337 (-22.88%)	26.4	179 (-36.97%)	284 (-35.01%)	25.31
graph_1	**390.8 (3.54)**	394	600	206 (-20.77%)	260 (-33.47%)	16.85	184 (-21.03%)	233 (-40.38%)	16.74
graph_2	**412.0 (2.76)**	417	600	282 (-8.14%)	307 (-25.49%)	30.48	169 (-34.5%)	258 (-37.38%)	30.58
graph_3	**374.6 (2.65)**	378	600	252 (-8.36%)	275 (-26.59%)	15.89	181 (-27.31%)	249 (-33.53%)	15.75
graph_4	**498.2 (1.17)**	500	600	241 (-25.39%)	323 (-35.17%)	149.6	158 (-32.48%)	234 (-53.03%)	146.36
graph_5	**483.6 (4.59)**	490	600	313 (-14.71%)	367 (-24.11%)	48.77	209 (-35.09%)	322 (-33.42%)	48.59
graph_6	**395.0 (2.97)**	398	600	218 (-24.04%)	287 (-27.34%)	25.78	138 (-45.67%)	254 (-35.7%)	25.74
graph_7	**364.8 (2.14)**	369	600	277 (-6.1%)	295 (-19.13%)	11.94	163 (-40.07%)	272 (-25.44%)	12.03
graph_8	**413.8 (2.93)**	418	600	260 (-8.45%)	284 (-31.37%)	35.47	176 (-35.53%)	273 (-34.03%)	35.46
graph_9	**415.4 (2.42)**	418	600	257 (0.0%)	257 (-38.13%)	22.56	209 (-18.99%)	258 (-37.89%)	22.66
graph_10	**414.4 (5.2)**	420	600	259 (-5.47%)	274 (-33.88%)	18.26	227 (-12.02%)	258 (-37.74%)	18.43
graph_11	**434.2 (3.82)**	440	600	215 (-24.83%)	286 (-34.13%)	58.39	134 (-40.71%)	226 (-47.95%)	58.14
graph_12	**372.8 (5.49)**	382	600	204 (-15.0%)	240 (-35.62%)	14.42	216 (-17.24%)	261 (-29.99%)	14.48
graph_13	**481.2 (2.32)**	484	600	210 (-40.0%)	350 (-27.27%)	37.4	387 (-3.73%)	402 (-16.46%)	37.76
graph_14	**353.6 (2.42)**	358	600	236 (-2.88%)	243 (-31.28%)	6.72	137 (-35.98%)	214 (-39.48%)	6.74

Table 5.25: The results for $k = 10$. **DCCA-Same**: mean difference between TOPKWCLQ is -29.72% (SD = 5.48%) with the mean difference between the end and the max score being -14.62% (SD = 10.75%). **DCCA-Mix**: mean difference between TOPKWCLQ is -35.83% (SD = 8.53%) with the mean difference between the end and the max score being -29.15% (SD = 11.96%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**875.2 (4.62)**	881	600	492 (-6.29%)	525 (-40.01%)	38.21	479 (-20.3%)	601 (-31.33%)	39.32
graph_1	**862.4 (6.15)**	874	600	528 (-7.85%)	573 (-33.56%)	25.56	533 (-6.65%)	571 (-33.79%)	25.86
graph_2	**835.8 (1.94)**	838	600	502 (-7.04%)	540 (-35.39%)	48.17	495 (-11.92%)	562 (-32.76%)	46.42
graph_3	**782.2 (3.82)**	786	600	521 (-5.96%)	554 (-29.17%)	23.71	507 (-16.06%)	604 (-22.78%)	24.02
graph_4	**936.8 (1.72)**	939	600	326 (-23.29%)	425 (-54.63%)	228.42	453 (-18.23%)	554 (-40.86%)	226.67
graph_5	**919.6 (3.98)**	923	600	528 (-5.55%)	559 (-39.21%)	75.07	506 (-16.91%)	609 (-33.78%)	74.69
graph_6	**886.8 (4.26)**	894	600	449 (-17.77%)	546 (-38.43%)	39.52	448 (-18.25%)	548 (-38.2%)	39.15
graph_7	**810.8 (2.48)**	814	600	472 (-20.27%)	592 (-26.99%)	18.28	481 (-17.92%)	586 (-27.73%)	18.25
graph_8	**831.8 (5.98)**	842	600	511 (-12.95%)	587 (-29.43%)	53.04	503 (-9.37%)	555 (-33.28%)	53.97
graph_9	**821.6 (2.33)**	824	600	629 (-3.97%)	655 (-20.28%)	34.21	326 (-2.98%)	336 (-59.1%)	34.68
graph_10	**870.8 (3.71)**	876	600	534 (-10.4%)	596 (-31.56%)	27.81	525 (-11.32%)	592 (-32.02%)	27.78
graph_11	**898.8 (6.24)**	910	600	501 (-7.39%)	541 (-39.81%)	88.7	472 (-12.1%)	537 (-40.25%)	90.3
graph_12	**810.8 (2.23)**	813	600	481 (-17.21%)	581 (-28.34%)	21.77	528 (-7.53%)	571 (-29.58%)	21.97
graph_13	**870.4 (4.22)**	875	600	290 (-10.22%)	323 (-62.89%)	57.13	521 (0.0%)	521 (-40.14%)	57.03
graph_14	**776.4 (5.75)**	787	600	566 (-5.03%)	596 (-23.24%)	10.24	294 (-14.29%)	343 (-55.82%)	10.25

Table 5.26: The results for $k = 30$. **DCCA-Same**: mean difference between TOPKWCLQ is -35.53% (SD = 11.27%) with the mean difference between the end and the max score being -10.75% (SD = 6.13%). **DCCA-Mix**: mean difference between TOPKWCLQ is -36.76% (SD = 9.75%) with the mean difference between the end and the max score being -12.26% (SD = 6.02%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**1263.0 (6.72)**	1274	600	850 (-0.82%)	857 (-32.15%)	51.16	798 (-4.32%)	834 (-33.97%)	52.32
graph_1	**1269.6 (10.13)**	1289	600	874 (-2.35%)	895 (-29.51%)	34.48	863 (-7.7%)	935 (-26.35%)	35.65
graph_2	**1222.0 (4.98)**	1229	600	733 (-8.26%)	799 (-34.62%)	61.54	724 (-14.32%)	845 (-30.85%)	61.82
graph_3	**1161.2 (2.64)**	1164	600	798 (-6.01%)	849 (-26.89%)	31.7	837 (-2.9%)	862 (-25.77%)	32.74
graph_4	**1323.4 (4.41)**	1330	600	196 (-49.87%)	391 (-70.45%)	303.5	761 (-12.33%)	868 (-34.41%)	302.58
graph_5	**1300.0 (4.94)**	1306	600	814 (-0.49%)	818 (-37.08%)	100.41	849 (-5.03%)	894 (-31.23%)	100.3
graph_6	**1314.2 (3.71)**	1318	600	811 (-5.7%)	860 (-34.56%)	52.54	786 (-6.54%)	841 (-36.01%)	52.56
graph_7	**1204.2 (3.76)**	1209	600	778 (-9.43%)	859 (-28.67%)	24.35	817 (-1.92%)	833 (-30.83%)	24.9
graph_8	**1213.6 (3.83)**	1219	600	601 (-5.5%)	636 (-47.59%)	71.39	791 (-5.83%)	840 (-30.78%)	73.84
graph_9	**1190.6 (1.96)**	1193	600	900 (0.0%)	900 (-24.41%)	45.65	662 (-1.63%)	673 (-43.47%)	47.53
graph_10	**1268.0 (3.95)**	1274	600	814 (-3.78%)	846 (-33.28%)	37.19	834 (-7.33%)	900 (-29.02%)	35.55
graph_11	**1294.0 (8.41)**	1308	600	229 (-37.6%)	367 (-71.64%)	118.66	802 (-6.85%)	861 (-33.46%)	117.75
graph_12	**1203.4 (2.87)**	1207	600	791 (-9.81%)	877 (-27.12%)	29.73	852 (-2.41%)	873 (-27.46%)	29.85
graph_13	**1230.2 (3.43)**	1235	600	261 (-27.5%)	360 (-70.74%)	74.76	920 (0.0%)	920 (-25.22%)	76.02
graph_14	**1154.8 (6.4)**	1165	600	883 (-0.9%)	891 (-22.84%)	13.56	795 (0.0%)	795 (-31.16%)	13.87

Table 5.27: The results for $k = 50$. **DCCA-Same**: mean difference between TOPKWCLQ is -39.44% (SD = 17.35%) with the mean difference between the end and the max score being -11.2% (SD = 15.0%). **DCCA-Mix**: mean difference between TOPKWCLQ is -31.33% (SD = 4.66%) with the mean difference between the end and the max score being -5.27% (SD = 4.15%).

TOPKWCLQ cutoff of 60 seconds

When we combine the results of all the three tested values of k (tables 5.28, 5.29, and 5.30) with a cutoff of 60 seconds, we see that the mean difference between *TOPKWCLQ* and DCCA-Same is -34.12% (SD = 11.51) and the mean difference between *TOPKWCLQ* and DCCA-Mix is -33.89% (SD = 7.75%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**431.6 (2.87)**	436	60	286 (-15.13%)	337 (-21.92%)	26.4	179 (-36.97%)	284 (-34.2%)	25.31
graph_1	**384.2 (2.48)**	389	60	206 (-20.77%)	260 (-32.33%)	16.85	184 (-21.03%)	233 (-39.35%)	16.74
graph_2	**407.2 (5.91)**	417	60	282 (-8.14%)	307 (-24.61%)	30.48	169 (-34.5%)	258 (-36.64%)	30.58
graph_3	**368.8 (4.07)**	374	60	252 (-8.36%)	275 (-25.43%)	15.89	181 (-27.31%)	249 (-32.48%)	15.75
graph_4	**490.4 (3.44)**	495	60	241 (-25.39%)	323 (-34.14%)	149.6	158 (-32.48%)	234 (-52.28%)	146.36
graph_5	**479.0 (7.92)**	489	60	313 (-14.71%)	367 (-23.38%)	48.77	209 (-35.09%)	322 (-32.78%)	48.59
graph_6	**386.0 (3.85)**	392	60	218 (-24.04%)	287 (-25.65%)	25.78	138 (-45.67%)	254 (-34.2%)	25.74
graph_7	**357.8 (3.43)**	361	60	277 (-6.1%)	295 (-17.55%)	11.94	163 (-40.07%)	272 (-23.98%)	12.03
graph_8	**404.6 (4.08)**	412	60	260 (-8.45%)	284 (-29.81%)	35.47	176 (-35.53%)	273 (-32.53%)	35.46
graph_9	**411.0 (3.35)**	417	60	257 (0.0%)	257 (-37.47%)	22.56	209 (-18.99%)	258 (-37.23%)	22.66
graph_10	**410.8 (6.55)**	420	60	259 (-5.47%)	274 (-33.3%)	18.26	227 (-12.02%)	258 (-37.2%)	18.43
graph_11	**425.6 (4.41)**	433	60	215 (-24.83%)	286 (-32.8%)	58.39	134 (-40.71%)	226 (-46.9%)	58.14
graph_12	**365.4 (6.15)**	376	60	204 (-15.0%)	240 (-34.32%)	14.42	216 (-17.24%)	261 (-28.57%)	14.48
graph_13	**475.0 (4.0)**	481	60	210 (-40.0%)	350 (-26.32%)	37.4	387 (-3.73%)	402 (-15.37%)	37.76
graph_14	**348.8 (1.72)**	352	60	236 (-2.88%)	243 (-30.33%)	6.72	137 (-35.98%)	214 (-38.65%)	6.74

Table 5.28: The results for $k = 10$. **DCCA-Same**: mean difference between TOPKQ-CLQ is -28.62% (SD = 5.57%) with the mean difference between the end and the max score being -14.62% (SD = 10.75%). **DCCA-Mix**: mean difference between TOPKQ-CLQ is -34.82% (SD = 8.67%) with the mean difference between the end and the max score being -29.15% (SD = 11.96%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**861.0 (3.9)**	866	60	492 (-6.29%)	525 (-39.02%)	38.21	479 (-20.3%)	601 (-30.2%)	39.32
graph_1	**856.8 (5.27)**	866	60	528 (-7.85%)	573 (-33.12%)	25.56	533 (-6.65%)	571 (-33.36%)	25.86
graph_2	**827.0 (6.36)**	835	60	502 (-7.04%)	540 (-34.7%)	48.17	495 (-11.92%)	562 (-32.04%)	46.42
graph_3	**774.8 (4.96)**	784	60	521 (-5.96%)	554 (-28.5%)	23.71	507 (-16.06%)	604 (-22.04%)	24.02
graph_4	**931.8 (5.27)**	939	60	326 (-23.29%)	425 (-54.39%)	228.42	453 (-18.23%)	554 (-40.55%)	226.67
graph_5	**909.0 (4.94)**	916	60	528 (-5.55%)	559 (-38.5%)	75.07	506 (-16.91%)	609 (-33.0%)	74.69
graph_6	**876.0 (4.05)**	882	60	449 (-17.77%)	546 (-37.67%)	39.52	448 (-18.25%)	548 (-37.44%)	39.15
graph_7	**800.8 (3.71)**	805	60	472 (-20.27%)	592 (-26.07%)	18.28	481 (-17.92%)	586 (-26.82%)	18.25
graph_8	**821.8 (3.31)**	826	60	511 (-12.95%)	587 (-28.57%)	53.04	503 (-9.37%)	555 (-32.47%)	53.97
graph_9	**814.8 (4.71)**	822	60	629 (-3.97%)	655 (-19.61%)	34.21	326 (-2.98%)	336 (-58.76%)	34.68
graph_10	**859.4 (3.93)**	865	60	534 (-10.4%)	596 (-30.65%)	27.81	525 (-11.32%)	592 (-31.11%)	27.78
graph_11	**888.0 (3.35)**	894	60	501 (-7.39%)	541 (-39.08%)	88.7	472 (-12.1%)	537 (-39.53%)	90.3
graph_12	**803.8 (5.15)**	808	60	481 (-17.21%)	581 (-27.72%)	21.77	528 (-7.53%)	571 (-28.96%)	21.97
graph_13	**864.4 (1.02)**	866	60	290 (-10.22%)	323 (-62.63%)	57.13	521 (0.0%)	521 (-39.73%)	57.03
graph_14	**764.4 (1.02)**	766	60	566 (-5.03%)	596 (-22.03%)	10.24	294 (-14.29%)	343 (-55.13%)	10.25

Table 5.29: The results for $k = 30$. **DCCA-Same**: mean difference between TOPKQ-CLQ is -34.82% (SD = 11.45%) with the mean difference between the end and the max score being -10.75% (SD = 6.13%). **DCCA-Mix**: mean difference between TOPKQ-CLQ is -36.08% (SD = 9.87%) with the mean difference between the end and the max score being -12.26% (SD = 6.02%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
graph_0	**1248.8 (5.04)**	1255	60	850 (-0.82%)	857 (-31.37%)	51.16	798 (-4.32%)	834 (-33.22%)	52.32
graph_1	**1253.8 (10.09)**	1270	60	874 (-2.35%)	895 (-28.62%)	34.48	863 (-7.7%)	935 (-25.43%)	35.65
graph_2	**1215.8 (6.73)**	1226	60	733 (-8.26%)	799 (-34.28%)	61.54	724 (-14.32%)	845 (-30.5%)	61.82
graph_3	**1151.6 (3.5)**	1158	60	798 (-6.01%)	849 (-26.28%)	31.7	837 (-2.9%)	862 (-25.15%)	32.74
graph_4	**1313.8 (8.77)**	1330	60	196 (-49.87%)	391 (-70.24%)	303.5	761 (-12.33%)	868 (-33.93%)	302.58
graph_5	**1290.2 (7.49)**	1302	60	814 (-0.49%)	818 (-36.6%)	100.41	849 (-5.03%)	894 (-30.71%)	100.3
graph_6	**1306.4 (6.62)**	1316	60	811 (-5.7%)	860 (-34.17%)	52.54	786 (-6.54%)	841 (-35.62%)	52.56
graph_7	**1192.4 (4.27)**	1198	60	778 (-9.43%)	859 (-27.96%)	24.35	817 (-1.92%)	833 (-30.14%)	24.9
graph_8	**1204.0 (2.61)**	1206	60	601 (-5.5%)	636 (-47.18%)	71.39	791 (-5.83%)	840 (-30.23%)	73.84
graph_9	**1183.4 (6.28)**	1191	60	900 (0.0%)	900 (-23.95%)	45.65	662 (-1.63%)	673 (-43.13%)	47.53
graph_10	**1258.4 (5.12)**	1263	60	814 (-3.78%)	846 (-32.77%)	37.19	834 (-7.33%)	900 (-28.48%)	35.55
graph_11	**1281.2 (5.34)**	1290	60	229 (-37.6%)	367 (-71.35%)	118.66	802 (-6.85%)	861 (-32.8%)	117.75
graph_12	**1195.2 (4.75)**	1203	60	791 (-9.81%)	877 (-26.62%)	29.73	852 (-2.41%)	873 (-26.96%)	29.85
graph_13	**1223.6 (0.8)**	1225	60	261 (-27.5%)	360 (-70.58%)	74.76	920 (0.0%)	920 (-24.81%)	76.02
graph_14	**1140.4 (3.88)**	1145	60	883 (-0.9%)	891 (-21.87%)	13.56	795 (0.0%)	795 (-30.29%)	13.87

Table 5.30: The results for $k = 50$. **DCCA-Same**: mean difference between TOPKQ-CLQ is -38.92% (SD = 17.52%) with the mean difference between the end and the max score being -11.2% (SD = 15.0%). **DCCA-Mix**: mean difference between TOPKQ-CLQ is -30.76% (SD = 4.73%) with the mean difference between the end and the max score being -5.27% (SD = 4.15%).

Results T-Test

The results of the dependent T-test between the percentage difference of *TOPKWCLQ* with a cutoff time of 600 seconds and 60 seconds for both DCCA-Same and DCCA-Mix are as follow:

- If $k = 10$, then for DCCA-Same, the difference between a cutoff time of 60 seconds ($M = -28.62\%$, $SD = 5.57\%$) and 600 seconds ($M = -29.72\%$, $SD = 5.48\%$) is significant, $T(14)=12.5459$, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds ($M = -34.82\%$, $SD = 8.67\%$) or 600 seconds ($M = -35.83\%$, $SD = 8.53\%$) is significant, $T(14)=11.7241$, $p < 0.01$.

- If $k = 30$, then for DCCA-Same, the difference between a cutoff time of 60 seconds ($M = -34.82\%$, $SD = 11.45\%$) and 600 seconds ($M = -35.53\%$, $SD = 11.27\%$) is significant, $T(14)=10.6381$, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds ($M = -36.08\%$, $SD = 9.87\%$) or 600 seconds ($M = -36.76\%$, $SD = 9.75\%$) is significant, $T(14)=11.6640$, $p < 0.01$.

- If $k = 50$, then for DCCA-Same, the difference between a cutoff time of 60 seconds ($M = -38.92\%$, $SD = 17.52\%$) and 600 seconds ($M = -39.44\%$, $SD = 17.35\%$) is significant, $T(14)=8.3399$, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds ($M = -30.76\%$, $SD = 4.73\%$) or 600 seconds ($M = -31.33\%$, $SD = 4.66\%$) is significant, $T(14)=12.3291$, $p < 0.01$.

5.2.3 Real-world Graphs

This difference between DCCA and *TOPKWCLQ* is even more noticeable in the real-world graphs, shown in tables 5.31, 5.32, and 5.33. These tables show again that for

the graphs that DCCA finished, DCCA-Mix scores better than DCCA-Same for $k = 50$ and DCCA-Same scores better with $k = 10$ and $k = 30$.

TOPKWCLQ cutoff of 600 seconds

When we combine the results of all the three tested values of k (tables 5.31, 5.32, and 5.33), we see that for all the graphs, the mean difference between *TOPKWCLQ* and DCCA-Same is -47.37% (SD = 26.2) and the mean difference between *TOPKWCLQ* and DCCA-Mix is -48.33% (SD = 25.03%). The mean difference between the end and the max score for DCCA-Same is -16.39% (SD = 16.93%) and for DCCA-Mix is -16.79% (SD = 13.68%).

For the graphs that DCCA finished, the mean difference between *TOPKWCLQ* and DCCA-Same is -36.29% (SD = 19.94) and the mean difference between *TOPKWCLQ* and DCCA-Mix is -37.25% (SD = 19.86%). The mean difference between the end and the max score for DCCA-Same is -13.09% (SD = 16.3%) and for DCCA-Mix is -11.66% (SD = 11.44%).

Lastly, for the graphs that DCCA timed out at, the mean difference between *TOPKWCLQ* and DCCA-Same is -69.54% (SD = 23.3) and the mean difference between *TOPKWCLQ* and DCCA-Mix is -70.49% (SD = 19.26%). The mean difference between the end and the max score for DCCA-Same is -23% (SD = 15.61%) and for DCCA-Mix is -27.06% (SD = 11.72%).

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
ca-GrQc	**1481.0 (0.0)**	1481	600	1142 (-8.57%)	1249 (-15.67%)	3.83	1142 (-7.83%)	1239 (-16.34%)	3.95
ca-netscience	**391.0 (0.0)**	391	600	352 (0.0%)	352 (-9.97%)	0.15	330 (-1.79%)	336 (-14.07%)	0.15
ia-email-univ	**444.8 (1.47)**	447	600	158 (-19.8%)	197 (-55.71%)	2.78	158 (-18.13%)	193 (-56.61%)	2.57
ia-infect-dublin	**622.0 (0.0)**	622	600	370 (-0.27%)	371 (-40.35%)	1.08	231 (-3.75%)	240 (-61.41%)	1.05
inf-power	**286.0 (0.0)**	286	600	196 (0.0%)	196 (-31.47%)	4.72	162 (-12.43%)	185 (-35.31%)	4.71
rt-retweet	**198.0 (0.0)**	198	600	142 (0.0%)	142 (-28.28%)	0.07	130 (0.0%)	130 (-34.34%)	0.07
sc-shipsec1	**1377.0 (6.6)**	1389	600	1082 (-5.99%)	1151 (-16.41%)	**900.0**	560 (-37.08%)	890 (-35.37%)	**900.0**
soc-buzznet	**904.4 (15.29)**	914	600	90 (-52.63%)	190 (-78.99%)	**900.0**	169 (-33.73%)	255 (-71.8%)	**900.0**
socfb-CMU	**1772.0 (10.33)**	1790	600	238 (-27.44%)	328 (-81.49%)	**900.0**	211 (-35.67%)	328 (-81.49%)	**900.0**
tech-RL-caida	**683.2 (17.53)**	715	600	143 (-56.93%)	332 (-51.41%)	**900.0**	141 (-46.79%)	265 (-61.21%)	**900.0**
tech-WHOIS	**1626.6 (12.08)**	1640	600	309 (-5.21%)	326 (-79.96%)	**900.0**	316 (-3.36%)	327 (-79.9%)	**900.0**
tech-internet-as	**391.8 (11.72)**	413	600	190 (-17.75%)	231 (-41.04%)	90.64	267 (-11.59%)	302 (-22.92%)	90.87
tech-routers-rf	**585.0 (3.16)**	589	600	266 (-0.37%)	267 (-54.36%)	2.94	225 (0.0%)	225 (-61.54%)	2.89
web-arabic-2005	**4049.0 (0.0)**	4049	600	2706 (-7.55%)	2927 (-27.71%)	365.21	2098 (-12.07%)	2386 (-41.07%)	357.55
web-spam	**720.2 (2.48)**	723	600	218 (-41.55%)	373 (-48.21%)	55.82	218 (-36.26%)	342 (-52.51%)	55.37

Table 5.31: The results for $k = 10$. If the runtime for DCCA is stated in bold, it means that DCCA terminated early on those graphs, because it exceeded the 900 seconds timeout limit. **DCCA-Same**: mean difference between TOPKWCLQ is -44.07% (SD = 23.38%) for all graphs, -35.28% (SD = 15.47%) for the finished graphs and -61.65% (SD = 28.2%) for the not finished graphs and with the mean difference between the end and the max score being -16.27% (SD = 19.68%) for all graphs, -9.59% (SD = 13.52%) for the finished graphs and -29.64% (SD = 24.67%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKWCLQ is -48.39% (SD = 21.83%) for all graphs, -39.61% (SD = 18.05%) for the finished graphs and -65.95% (SD = 18.89%) for the not finished graphs with the mean difference between the end and the max score being -17.37% (SD = 16.08%) for all graphs, -10.38% (SD = 10.94%) for the finished graphs and -31.33% (SD = 16.42%) for the not finished graphs.

	TOPKWCLQ			DCCA-Same			DCCA-Mix		
Graphs	**Result**	**Max**	**Run Time**	**End Score**	**Max Score**	**Run Time**	**End Score**	**Max Score**	**Run Time**
ca-GrQc	**2511.2 (1.47)**	2513	600	2220 (-3.18%)	2293 (-8.69%)	5.94	2220 (-3.18%)	2293 (-8.69%)	6.1
ca-netscience	**947.6 (0.8)**	948	600	808 (-0.62%)	813 (-14.2%)	0.21	798 (-1.97%)	814 (-14.1%)	0.21
ia-email-univ	**1020.8 (1.47)**	1022	600	430 (-12.24%)	490 (-52.0%)	3.99	422 (-11.9%)	479 (-53.08%)	4.05
ia-infect-dublin	**1269.6 (3.93)**	1276	600	531 (-13.8%)	616 (-51.48%)	1.52	521 (-9.55%)	576 (-54.63%)	1.51
inf-power	**775.8 (0.75)**	777	600	590 (0.0%)	590 (-23.95%)	7.31	419 (-12.34%)	478 (-38.39%)	7.34
rt-retweet	**434.0 (0.0)**	434	600	350 (0.0%)	350 (-19.35%)	0.08	202 (-25.19%)	270 (-37.79%)	0.08
sc-shipsec1	**3797.2 (16.07)**	3816	600	1740 (-30.2%)	2493 (-34.35%)	**900.0**	1771 (-29.47%)	2511 (-33.87%)	**900.0**
soc-buzznet	**2016.6 (27.36)**	2058	600	227 (-21.18%)	288 (-85.72%)	**900.0**	193 (-39.69%)	320 (-84.13%)	**900.0**
socfb-CMU	**3931.0 (14.18)**	3951	600	463 (-7.95%)	503 (-87.2%)	**900.0**	517 (-28.19%)	720 (-81.68%)	**900.0**
tech-RL-caida	**1561.0 (15.71)**	1581	600	438 (-11.52%)	495 (-68.29%)	**900.0**	364 (-31.06%)	528 (-66.18%)	**900.0**
tech-WHOIS	**2807.8 (12.81)**	2830	600	376 (-32.86%)	560 (-80.06%)	**900.0**	345 (-21.23%)	438 (-84.4%)	**900.0**
tech-internet-as	**884.8 (7.65)**	897	600	181 (-53.35%)	388 (-56.15%)	135.17	390 (-34.12%)	592 (-33.09%)	140.35
tech-routers-rf	**1167.0 (3.46)**	1172	600	353 (-23.43%)	461 (-60.5%)	4.38	420 (-11.39%)	474 (-59.38%)	4.43
web-arabic-2005	**10483.0 (0.0)**	10483	600	7240 (-7.08%)	7792 (-25.67%)	564.19	9099 (-1.21%)	9210 (-12.14%)	600.55
web-spam	**1589.4 (7.96)**	1597	600	369 (-44.18%)	661 (-58.41%)	85.25	369 (-42.25%)	639 (-59.8%)	86.33

Table 5.32: The results for $k = 30$. If the runtime for DCCA is stated in bold, it means that DCCA terminated early on those graphs, because it exceeded the 900 seconds timeout limit. **DCCA-Same**: mean difference between TOPKWCLQ is -48.4% (SD = 26.09%) for all graphs, -37.04% (SD = 20.39%) for the finished graphs and -71.12% (SD = 21.86%) for the not finished graphs and with the mean difference between the end and the max score being -17.44% (SD = 16.52%) for all graphs, -15.79% (SD = 19.02%) for the finished graphs and -20.74% (SD = 11.01%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKWCLQ is -48.09% (SD = 25.4%) for all graphs, -37.11% (SD = 19.86%) for the finished graphs and -70.05% (SD = 21.58%) for the not finished graphs with the mean difference between the end and the max score being -20.18% (SD = 13.78%) for all graphs, -15.31% (SD = 14.03%) for the finished graphs and -29.93% (SD = 6.62%) for the not finished graphs.

Graphs	TOPKWCLQ Result	TOPKWCLQ Max	TOPKWCLQ Run Time	DCCA-Same End Score	DCCA-Same Max Score	DCCA-Same Run Time	DCCA-Mix End Score	DCCA-Mix Max Score	DCCA-Mix Run Time
ca-GrQc	**3257.0 (5.9)**	3264	600	2930 (-2.27%)	2998 (-7.95%)	8.13	2893 (-2.79%)	2976 (-8.63%)	8.36
ca-netscience	**1304.2 (0.75)**	1305	600	1150 (0.0%)	1150 (-11.82%)	0.25	1128 (-0.79%)	1137 (-12.82%)	0.27
ia-email-univ	**1490.6 (3.98)**	1496	600	724 (-0.69%)	729 (-51.09%)	5.24	677 (-2.45%)	694 (-53.44%)	5.78
ia-infect-dublin	**1662.0 (1.26)**	1664	600	665 (-11.45%)	751 (-54.81%)	2.02	665 (-11.21%)	749 (-54.93%)	2.0
inf-power	**1220.0 (2.28)**	1224	600	743 (-13.2%)	856 (-29.84%)	10.45	743 (-11.86%)	843 (-30.9%)	9.87
rt-retweet	**539.0 (0.0)**	539	600	447 (-3.25%)	462 (-14.29%)	0.08	423 (-2.98%)	436 (-19.11%)	0.08
sc-shipsec1	**6047.8 (23.37)**	6079	600	2960 (-14.6%)	3466 (-42.69%)	**900.0**	3084 (-5.28%)	3256 (-46.16%)	**900.0**
soc-buzznet	**2959.6 (36.27)**	3012	600	227 (-26.77%)	310 (-89.53%)	**900.0**	235 (-27.91%)	326 (-88.98%)	**900.0**
socfb-CMU	**5490.4 (19.16)**	5515	600	442 (-15.16%)	521 (-90.51%)	**900.0**	784 (-27.0%)	1074 (-80.44%)	**900.0**
tech-RL-caida	**2346.4 (15.37)**	2361	600	615 (-4.06%)	641 (-72.68%)	**900.0**	541 (-8.31%)	590 (-74.86%)	**900.0**
tech-WHOIS	**3590.6 (18.65)**	3621	600	394 (-32.42%)	583 (-83.76%)	**900.0**	326 (-31.08%)	473 (-86.83%)	**900.0**
tech-internet-as	**1349.6 (6.71)**	1362	600	231 (-50.0%)	462 (-65.77%)	186.18	697 (-25.53%)	936 (-30.65%)	196.25
tech-routers-rf	**1630.6 (4.96)**	1635	600	488 (-27.6%)	674 (-58.67%)	5.77	492 (-25.57%)	661 (-59.46%)	6.42
web-arabic-2005	**16637.0 (0.0)**	16637	600	14480 (-3.29%)	14972 (-10.01%)	802.15	13860 (-2.95%)	14282 (-14.16%)	799.37
web-spam	**2271.2 (7.88)**	2283	600	640 (-27.36%)	881 (-61.21%)	115.58	715 (-6.78%)	767 (-66.23%)	121.69

Table 5.33: The results for $k = 50$. If the runtime for DCCA is stated in bold, it means that DCCA terminated early on those graphs, because it exceeded the 900 seconds time-out limit. **DCCA-Same**: mean difference between TOPKWCLQ is -49.64% (SD = 29.14%) for all graphs, -36.55% (SD = 23.97%) for the finished graphs and -75.83% (SD = 19.84%) for the not finished graphs and with the mean difference between the end and the max score being -15.47% (SD = 14.59%) for all graphs, -13.91% (SD = 16.36%) for the finished graphs and -18.6% (SD = 11.15%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKWCLQ is -48.51% (SD = 27.86%) for all graphs, -35.03% (SD = 21.66%) for the finished graphs and -75.45% (SD = 17.29%) for the not finished graphs with the mean difference between the end and the max score being -12.83% (SD = 11.18%) for all graphs, -9.29% (SD = 9.35%) for the finished graphs and -19.92% (SD = 12.12%) for the not finished graphs.

TOPKWCLQ cutoff of 60 seconds

When we combine the results of all the three tested values of k (tables 5.34, 5.35, and 5.36) for a cutoff of 60 seconds, we see that for all the graphs, the mean difference between *TOPKWCLQ* and DCCA-Same is -46.95% (SD = 26.1%) and the mean difference between *TOPKWCLQ* and DCCA-Mix is -47.86% (SD = 25.09%).

For the graphs that DCCA finished, the mean difference between *TOPKWCLQ* and DCCA-Same is -35.92% (SD = 19.76%) and the mean difference between *TOPKWCLQ* and DCCA-Mix is -36.81% (SD = 19.99%).

Lastly, for the graphs that DCCA timed out at, the mean difference between *TOPKWCLQ* and DCCA-Same is -69.02% (SD = 23.41%) and the mean difference between *TOPKWCLQ* and DCCA-Mix is -69.97% (SD = 19.33%).

Graphs	TOPKWCLQ			DCCA-Same			DCCA-Mix		
	Result	Max	Run Time	End Score	Max Score	Run Time	End Score	Max Score	Run Time
ca-GrQc	**1481.0 (0.0)**	1481	60	1142 (-8.57%)	1249 (-15.67%)	3.83	1142 (-7.83%)	1239 (-16.34%)	3.95
ca-netscience	**391.0 (0.0)**	391	60	352 (0.0%)	352 (-9.97%)	0.15	330 (-1.79%)	336 (-14.07%)	0.15
ia-email-univ	**440.6 (1.74)**	443	60	158 (-19.8%)	197 (-55.29%)	2.78	158 (-18.13%)	193 (-56.2%)	2.57
ia-infect-dublin	**622.0 (0.0)**	622	60	370 (-0.27%)	371 (-40.35%)	1.08	231 (-3.75%)	240 (-61.41%)	1.05
inf-power	**286.0 (0.0)**	286	60	196 (0.0%)	196 (-31.47%)	4.72	162 (-12.43%)	185 (-35.31%)	4.71
rt-retweet	**198.0 (0.0)**	198	60	142 (0.0%)	142 (-28.28%)	0.07	130 (0.0%)	130 (-34.34%)	0.07
sc-shipsec1	**1366.2 (4.26)**	1371	60	1082 (-5.99%)	1151 (-15.75%)	**900.0**	560 (-37.08%)	890 (-34.86%)	**900.0**
soc-buzznet	**847.8 (15.48)**	863	60	90 (-52.63%)	190 (-77.59%)	**900.0**	169 (-33.73%)	255 (-69.92%)	**900.0**
socfb-CMU	**1751.0 (24.73)**	1790	60	238 (-27.44%)	328 (-81.27%)	**900.0**	211 (-35.67%)	328 (-81.27%)	**900.0**
tech-RL-caida	**661.6 (13.63)**	678	60	143 (-56.93%)	332 (-49.82%)	**900.0**	141 (-46.79%)	265 (-59.95%)	**900.0**
tech-WHOIS	**1587.2 (14.03)**	1609	60	309 (-5.21%)	326 (-79.46%)	**900.0**	316 (-3.36%)	327 (-79.4%)	**900.0**
tech-internet-as	**367.4 (8.14)**	380	60	190 (-17.75%)	231 (-37.13%)	90.64	267 (-11.59%)	302 (-17.8%)	90.87
tech-routers-rf	**578.4 (4.32)**	582	60	266 (-0.37%)	267 (-53.84%)	2.94	225 (0.0%)	225 (-61.1%)	2.89
web-arabic-2005	**4047.0 (2.45)**	4049	60	2706 (-7.55%)	2927 (-27.67%)	365.21	2098 (-12.07%)	2386 (-41.04%)	357.55
web-spam	**715.6 (5.71)**	723	60	218 (-41.55%)	373 (-47.88%)	55.82	218 (-36.26%)	342 (-52.21%)	55.37

Table 5.34: The results for $k = 10$. **DCCA-Same**: mean difference between TOPKQCLQ is -43.43% (SD = 23.2%) for all graphs, -34.75% (SD = 15.19%) for the finished graphs and -60.78% (SD = 28.28%) for the not finished graphs and with the mean difference between the end and the max score being -16.27% (SD = 19.68%) for all graphs, -9.59% (SD = 13.52%) for the finished graphs and -29.64% (SD = 24.67%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKQCLQ is -47.68% (SD = 22.01%) for all graphs, -38.98% (SD = 18.51%) for the finished graphs and -65.08% (SD = 18.91%) for the not finished graphs with the mean difference between the end and the max score being -17.37% (SD = 16.08%) for all graphs, -10.38% (SD = 10.94%) for the finished graphs and -31.33% (SD = 16.42%) for the not finished graphs.

Graphs	TOPKWCLQ			DCCA-Same			DCCA-Mix		
	Result	Max	Run Time	End Score	Max Score	Run Time	End Score	Max Score	Run Time
ca-GrQc	**2505.4 (4.96)**	2512	60	2220 (-3.18%)	2293 (-8.48%)	5.94	2220 (-3.18%)	2293 (-8.48%)	6.1
ca-netscience	**945.6 (2.06)**	948	60	808 (-0.62%)	813 (-14.02%)	0.21	798 (-1.97%)	814 (-13.92%)	0.21
ia-email-univ	**1017.6 (3.93)**	1022	60	430 (-12.24%)	490 (-51.85%)	3.99	422 (-11.9%)	479 (-52.93%)	4.05
ia-infect-dublin	**1262.6 (1.85)**	1266	60	531 (-13.8%)	616 (-51.21%)	1.52	521 (-9.55%)	576 (-54.38%)	1.51
inf-power	**772.8 (0.75)**	774	60	590 (0.0%)	590 (-23.65%)	7.31	419 (-12.34%)	478 (-38.15%)	7.34
rt-retweet	**434.0 (0.0)**	434	60	350 (0.0%)	350 (-19.35%)	0.08	202 (-25.19%)	270 (-37.79%)	0.08
sc-shipsec1	**3768.8 (15.66)**	3793	60	1740 (-30.2%)	2493 (-33.85%)	**900.0**	1771 (-29.47%)	2511 (-33.37%)	**900.0**
soc-buzznet	**1952.4 (21.96)**	1984	60	227 (-21.18%)	288 (-85.25%)	**900.0**	193 (-39.69%)	320 (-83.61%)	**900.0**
socfb-CMU	**3888.0 (14.17)**	3909	60	463 (-7.95%)	503 (-87.06%)	**900.0**	517 (-28.19%)	720 (-81.48%)	**900.0**
tech-RL-caida	**1531.4 (16.7)**	1551	60	438 (-11.52%)	495 (-67.68%)	**900.0**	364 (-31.06%)	528 (-65.52%)	**900.0**
tech-WHOIS	**2766.6 (9.77)**	2775	60	376 (-32.86%)	560 (-79.76%)	**900.0**	345 (-21.23%)	438 (-84.17%)	**900.0**
tech-internet-as	**866.6 (11.48)**	886	60	181 (-53.35%)	388 (-55.23%)	135.17	390 (-34.12%)	592 (-31.69%)	140.35
tech-routers-rf	**1156.8 (8.63)**	1170	60	353 (-23.43%)	461 (-60.15%)	4.38	420 (-11.39%)	474 (-59.02%)	4.43
web-arabic-2005	**10476.2 (6.05)**	10483	60	7240 (-7.08%)	7792 (-25.62%)	564.19	9099 (-1.21%)	9210 (-12.09%)	600.55
web-spam	**1571.4 (13.75)**	1594	60	369 (-44.18%)	661 (-57.94%)	85.25	369 (-42.25%)	639 (-59.34%)	86.33

Table 5.35: The results for $k = 30$. **DCCA-Same**: mean difference between TOPKQCLQ is -48.07% (SD = 26.0%) for all graphs, -36.75% (SD = 20.24%) for the finished graphs and -70.72% (SD = 21.96%) for the not finished graphs and with the mean difference between the end and the max score being -17.44% (SD = 16.52%) for all graphs, -15.79% (SD = 19.02%) for the finished graphs and -20.74% (SD = 11.01%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKQCLQ is -47.73% (SD = 25.37%) for all graphs, -36.78% (SD = 19.82%) for the finished graphs and -69.63% (SD = 21.67%) for the not finished graphs with the mean difference between the end and the max score being -20.18% (SD = 13.78%) for all graphs, -15.31% (SD = 14.03%) for the finished graphs and -29.93% (SD = 6.62%) for the not finished graphs.

Graphs	TOPKWCLQ			DCCA-Same			DCCA-Mix		
	Result	Max	Run Time	End Score	Max Score	Run Time	End Score	Max Score	Run Time
ca-GrQc	**3245.6 (2.15)**	3249	60	2930 (-2.27%)	2998 (-7.63%)	8.13	2893 (-2.79%)	2976 (-8.31%)	8.36
ca-netscience	**1302.0 (0.89)**	1303	60	1150 (0.0%)	1150 (-11.67%)	0.25	1128 (-0.79%)	1137 (-12.67%)	0.27
ia-email-univ	**1480.6 (4.32)**	1486	60	724 (-0.69%)	729 (-50.76%)	5.24	677 (-2.45%)	694 (-53.13%)	5.78
ia-infect-dublin	**1655.8 (2.93)**	1659	60	665 (-11.45%)	751 (-54.64%)	2.02	665 (-11.21%)	749 (-54.77%)	2.0
inf-power	**1212.2 (2.04)**	1216	60	743 (-13.2%)	856 (-29.38%)	10.45	743 (-11.86%)	843 (-30.46%)	9.87
rt-retweet	**539.0 (0.0)**	539	60	447 (-3.25%)	462 (-14.29%)	0.08	423 (-2.98%)	436 (-19.11%)	0.08
sc-shipsec1	**5997.6 (19.06)**	6028	60	2960 (-14.6%)	3466 (-42.21%)	**900.0**	3084 (-5.28%)	3256 (-45.71%)	**900.0**
soc-buzznet	**2892.0 (47.68)**	2960	60	227 (-26.77%)	310 (-89.28%)	**900.0**	235 (-27.91%)	326 (-88.73%)	**900.0**
socfb-CMU	**5444.6 (36.1)**	5497	60	442 (-15.16%)	521 (-90.43%)	**900.0**	784 (-27.0%)	1074 (-80.27%)	**900.0**
tech-RL-caida	**2315.2 (23.58)**	2351	60	615 (-4.06%)	641 (-72.31%)	**900.0**	541 (-8.31%)	590 (-74.52%)	**900.0**
tech-WHOIS	**3557.4 (23.84)**	3589	60	394 (-32.42%)	583 (-83.61%)	**900.0**	326 (-31.08%)	473 (-86.7%)	**900.0**
tech-internet-as	**1320.8 (9.04)**	1331	60	231 (-50.0%)	462 (-65.02%)	186.18	697 (-25.53%)	936 (-29.13%)	196.25
tech-routers-rf	**1614.6 (10.8)**	1633	60	488 (-27.6%)	674 (-58.26%)	5.77	492 (-25.57%)	661 (-59.06%)	6.42
web-arabic-2005	**16628.4 (7.09)**	16637	60	14480 (-3.29%)	14972 (-9.96%)	802.15	13860 (-2.95%)	14282 (-14.11%)	799.37
web-spam	**2254.4 (5.89)**	2262	60	640 (-27.36%)	881 (-60.92%)	115.58	715 (-6.78%)	767 (-65.98%)	121.69

Table 5.36: The results for $k = 50$. **DCCA-Same**: mean difference between TOPKQCLQ is -49.36% (SD = 29.11%) for all graphs, -36.25% (SD = 23.84%) for the finished graphs and -75.57% (SD = 19.98%) for the not finished graphs and with the mean difference between the end and the max score being -15.47% (SD = 14.59%) for all graphs, -13.91% (SD = 16.36%) for the finished graphs and -18.6% (SD = 11.15%) for the not finished graphs. **DCCA-Mix**: mean difference between TOPKQCLQ is -48.18% (SD = 27.9%) for all graphs, -34.67% (SD = 21.63%) for the finished graphs and -75.19% (SD = 17.4%) for the not finished graphs with the mean difference between the end and the max score being -12.83% (SD = 11.18%) for all graphs, -9.29% (SD = 9.35%) for the finished graphs and -19.92% (SD = 12.12%) for the not finished graphs.

Results T-Test

The results of the dependent T-test between the percentage difference of *TOPKWCLQ* with a cutoff time of 600 seconds and 60 seconds for both DCCA-Same and DCCA-Mix are as follow:

- If $k = 10$, then for DCCA-Same, the difference between a cutoff time of 60 seconds (M = −43.43%, SD = 23.20%) and 600 seconds (M = −44.07%, SD = 23.38%) is insignificant, T(14)=2.3961, p=0.0311. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = −47.68, SD = 22.01%) or 600 seconds (M = −48.39%, SD = 21.83%) is insignificant, T(14)=2.0750, p=0.0569.
- If $k = 30$, then for DCCA-Same, the difference between a cutoff time of 60 seconds (M = −48.07%, SD = 26.00%) and 600 seconds (M = −48.40%, SD = 26.09%) is significant, T(14)=5.3263, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = −47.73%, SD = 25.37%) or 600 seconds (M = −48.09%, SD = 25.40%) is significant, T(14)=4.1122, $p < 0.01$.
- If $k = 50$, then for DCCA-Same, the difference between a cutoff time of 60 seconds (M = −49.36%, SD = 29.11%) and 600 seconds (M = −49.64%, SD = 29.14%) is significant, T(14)=5.5939, $p < 0.01$. For DCCA-Mix, the difference between a cutoff time of 60 seconds (M = −48.18%, SD = 27.90%) or 600 seconds (M = −48.51%, SD = 27.86%) is significant, T(14)=3.5953, $p < 0.01$.

Chapter 6

Discussion and Conclusion

Our discussion chapter will explain the results of our experiments, suggest future research and answer the research question. We start by discussing and explaining the results such that we can answer our research question. The following section will evaluate our main and sub research questions. Lastly, we will make recommendations for future research.

6.1 Discussion of the Results

Globally our results show that DCCA is currently not an improvement on the baselines, *TOPKLS* or *TOPKWCLQ*. However, we still see that it can be an improvement in some results (section 5.1.1) for the diversified top-k clique search problem (DTKC). This section will discuss four essential aspects of our results. The first aspect is the difference in score between DCCA and the baselines, *TOPKLS* and *TOPKWCLQ* and how it is depending on the evaluation graph set and the value of k. As previously stated, we saw DCCA only performed well on one evaluation graph set (table 4.5) and primarily for higher values of k. Our discussion of these results will explain why DCCA performed differently between the evaluation graph sets. Next, we will discuss the second aspect, namely, why the end and max scores differ in the results of DCCA.

The third aspect is the results of the runtime and its significance for the performance of DCCA and our research. We will explain what parts affected the runtime and what caused DCCA to vary so much in runtime between graphs. Moreover, we discuss the influence of the *encode-process-decode* paradigm on the runtime.

Lastly, we will discuss why DCCA performed significantly worse on the diversified top-k weighted clique search problem (DTKWC) than it did on DTKC. We will focus mainly on the reasons that only affected DCCA on DTKWC.

The results of the three evaluation graph sets for DTKC show that DCCA scored well on only one graph set, namely, the dual Barabási–Albert (BA) graph generated with the same input parameter. When comparing these graphs (table 4.5), we see that these graphs have significantly fewer maximal cliques than most real-world graphs (table 4.7) and the dual BA graphs generated by random input parameters (table 4.6). We believe

this indicates that DCCA has trouble learning the transition function. This inability to learn the transition function means that DCCA cannot learn which clique comes next from the Pivot Bron-Kerbosch algorithm (Tomita et al., 2006). Therefore, it does not know which cliques are already shown and which are still to come, which indicates that DCCA can not directly observe the complete state. Hence, DTKC and other diversity graphs problem should likely be formulated as a partially observable Markov decision process (POMDP) (Åström, Karl Johan, 1965), not as an MDP. The reason why DCCA performed better on smaller graphs is that the network architecture could better encode the transition function of those graphs, because they have less cliques. We believe this is the most crucial reason why DCCA did not perform well on larger graphs, with more cliques. We will provide possible solutions for this in our Future Research section.

However, this does not completely explain why DCCA performed worse on the smaller real-world graphs, such as *rt-retweet* and *ca-netscience*. The most likely cause for these results is that DCCA can not generalise well to graphs that are structurally different from those generated by the dual BA model. We expected this to happen; however, DCCA's inability to generalise is more substantial than expected. The previously stated problem of not observing the complete state, is likely the main reason for this and, therefore, our first reason. Nevertheless, this is likely not the only reason why it could not generalise well to differently structured graphs. We believe the second reason for these results is how we generated training graphs, which we only did through the dual BA model. This method is common for training deep RL methods for combinatorial optimisation problems. For example, the algorithm of Abe et al. (2019), on which we based our network design, also trained on generated graphs but their algorithm could generalise well to real-world graphs. However, their and all previous algorithms in the neural combinatorial optimisation with reinforcement learning (NCO-RL) research field are node-level CO problems, while DTKC is a subgraph-level task. Therefore, this might indicate that for subgraph-level CO problems, the training graphs should have more variation and can not be generated by a single model, if the goal is that the algorithm can generalise well between different graphs. Our Future Research section explains how future research could handle this problem.

We believe that the first reason, of not observing the whole state, and second reason, of how we generate training graphs, are the main causes for DCCA inability to generalise. However, there might another explanation why it could not generalise well. Our third reason why it could not generalise well between different graphs is that DCCA could likely not learn the maximum possible score of that graph and, therefore, could not compose the ideal clique set because it could not see how valuable a clique is for that given graph. For example, according to our analysis in table 4.7, *rt-retweet* and *ca-netscience* are similar in size in the number of nodes and edges. However, the coverage score found by *TOPKLS* differs significantly between the two graphs. For $k = 10$, the mean result for *rt-retweet* is 82 and for *ca-netscience* 222 (table 5.13). We already stated that this might be an issue in section 3.1.1 and that we could normalise the reward through the maximum clique for DTKC and the maximum weighted clique for DTKWC. However, these problems are NP-hard and would increase the training time significantly if we needed to find those cliques for each generated graph. Moreover, the values of those maximum cliques are likely not related to the maximum score of both DTKC and DTKWC. However, we are unsure about this being a problem, because most

NCO-RL approaches do not normalise the reward (Mazyavkina et al., 2021). This can indicate two things. First, it could mean that the problem of not normalising the reward is not an issue, and the first and second reasons explain the generalisation problem. Secondly, this normalisation problem is only relevant to diversity graph problems and not other CO graph problems. Therefore, we state this as our third reason because we are unsure if the normalisation is a problem. However, if future research can exclude the first and second reasons for the inability to generalise, it might be that the normalisation is the cause of the generalisation problem.

We also see a difference in results for DCCA between the different values of k, with DCCA performing better with higher values of k. We believe this primarily stems from two reasons. The first reason is that both *TOPKLS* and *TOPKWCLQ* can try fewer possible combinations with $k = 50$ than with $k = 10$. We know that number of possible formable clique sets in is the binomial coefficient of the number of cliques and k[1]. Therefore, there are significantly more combinations possible for $k = 50$. Because both *TOPKLS* and *TOPKWCLQ* terminate at a cutoff time, they have searched less of the possible solutions for higher values of k. For example, with $k = 10$, table 5.1 shows that the standard deviation of the results for *TOPKLS* for some graphs is 0. This likely indicates that the found result is the optimal clique set. Meanwhile, if $k = 50$, no standard deviation is 0 (table 5.3).

The second reason is that the actor and critic network have more input encodings with higher values of k. This increase in the number of inputs most likely benefits the critic network more than it does the actor network. In appendix B.1, we show the explained variance of each of our trained models[2]. The explained variance for when $k = 50$ is more stable then if $k = 10$. The difference in stability can also be caused by the clique set being more stable if $k = 50$ and thus more easily learned for the value network but we think that having more inputs is the more crucial reason. We believe this and the reason in the previous paragraph are the most likely reasons, with the first reason being more important, that DCCA performed better at higher values of k compared to the baseline algorithms, *TOPKLS* and *TOPKWCLQ*.

The previously stated problem of DCCA not observing the whole state likely also causes the difference between the max and end score for DCCA. This causes DCCA to not know if the best cliques are already shown or are still to come from the Pivot Bron-Kerbosch algorithm. Therefore, we believe that reformulating the MDP to a POMDP might solve this problem because it allows DCCA to learn which cliques are shown and which are not.

The results of the score show one interesting detail, namely, when comparing the two different trained versions of DCCA, DCCA-Mix and DCCA-Same, that DCCA-Mix almost always scored better on $k = 50$ and DCCA-Same on $k = 10$ and $k = 30$. We found these results to be too consistent to be a coincidence; therefore, there needs to be a reason why this happens. However, we could not find a direct cause for this. We believe that a reason might be that DCCA-Same gets overtrained on too similar graphs

[1]The number of possible clique sets is equal to $\binom{|C|}{k} = \frac{|C|!}{k!(|C|-k)!}$ in which $|C|$ is the number of maximal cliques in a graph.

[2]The explained variance indicates how much of the variance is explained by the model, with 1 being the highest number possible.

if $k = 50$ and prefers more variation in the training graphs at higher values of k and less with lower values of k. Nevertheless, we strongly believe that there are other factors that cause this result; therefore, more research is needed on why this occurred and what the most optimal method is to generate training graphs.

The results show a considerable variation in the runtime. This runtime mostly depends on the number of cliques in a graph. It shows that DCCA is considerably faster if the number of cliques is low. We see this mainly in the results of the dual BA graphs generated by the same input parameters in sections 5.1.1 and 5.2.1. This is especially important when $k = 50$ because DCCA outperforms *TOPKLS* on most graphs (tables 5.3 and 5.6). Therefore, we can clearly state that for $k = 50$, DCCA is the best method for this graph set for DTKC.

Nevertheless, it is also essential that this difference is significant when we lower the cutoff time. According to the T-test, the difference in results for when the cutoff time is 600 or 60 seconds is significant. Therefore, lowering the cutoff time only lets DCCA perform better in comparison to *TOPKLS* for this graph set. Moreover, the T-tests for the other graphs show similar results, with only two exceptions, namely for real-world graphs if $k = 10$, for both DTKC 5.13) and DTKWC (table 5.31). This means that the cutoff time has a significant factor on much the max score differs from the results of the baselines, *TOPKLS* and *TOPKWCLQ*, which indicates that DCCA is an improvement on runtime for smaller graphs.

The runtime is mainly affected by the number of cliques in a graph and, to a lesser extent, by the value of k and the graph size. That the number of cliques mainly affects the runtime is enormously significant for this research because it signifies that the clique finding algorithm causes the scalability issue of our algorithm and not the GNN architecture. We can state that this is because the runtime complexity of GIN is equal to the number of edges in the input graph (Wu et al., 2019), and we can see that the average number of steps per second for a graph does not scale at the same level. For example, if we look at the extended results in appendix C, we can see that when $k = 10$ for the graph *tech-internet-as*, the average number of steps per second is 785.35, and that for *ca-netscience*, it is 1371.43. This is still a difference of 74.63%; however, *tech-internet-as* has 85123 edges, and *ca-netscience* has 914 edges, which is a difference of 9213.24%. Therefore, using the *encode-process-decode* paradigm significantly helped with scaling our algorithm. Otherwise, we had to implement DCCA such that the whole graph acted as input at each step, meaning that the number of edges in a graph should affect the runtime significantly more than it does now. This result is significant because Cappart et al. (2021) state in their survey paper that one of the issues of using GIN architectures for graph combinatorial optimisation problems is the scalability issue to larger graphs. We showed that the *encode-process-decode* paradigm helps to overcome this issue. Therefore, the scalability issue is not caused by DCCA, but by the clique finding algorithm we used, the Pivot Bron-Kerbosch algorithm. This means that we only need to improve or change the clique finding algorithm, such that DCCA can scale to larger graphs in relation to the runtime.

Nevertheless, we see that the graph size affects the runtime. The most likely cause is a combination of how the runtime of the Pivot Bron-Kerbosch algorithm scales to larger graphs (Segundo et al., 2018) and the overhead within how we implemented DCCA, which are procedures like removing cliques from a set and storing graphs into memory.

The reason that larger graphs affect the runtime of the Pivot Bron-Kerbosch algorithm is that a larger percentage of the checked cliques will not be maximal. Therefore, it will run slower on larger graphs. We can prove that this influenced the runtime by comparing the runtimes of the same graph for different values of k and showing of different values of k affected the runtime.

For this example, we take graph_0 from the dual BA graphs generated with the same input parameters (table 4.5). In the results (section 5.1.1), we see that the runtime is 6.8 seconds, when $k = 10$, it is 10.52 seconds, when $k = 30$, and it is 13.44 seconds, when $k = 50$. Therefore, if the value of k had the biggest influence, then DCCA would run three times slower for $k = 30$ compared to $k = 10$. However, we only see a 54.71% increase in runtime. For $k = 50$, it would even be fives time longer, but the difference is 97.65%. Therefore, we can see that a significant part of the runtime is caused by finding the cliques in the graph. Still, we believe that the effect of k can be averted, which we will expand on in our Future Research section.

One of the most significant outcomes of this research is that DCCA scored significantly worse on the diversified top-k weighted clique search problem (DTKWC) than it did on DTKC. We believe the primary reason that DCCA scores lower on DTKWC is that the node's weights are over-smoothed in the output node encodings by the Graph Encoder. This over-smoothing is a common problem with GNN architectures (Chen et al., 2019). If over-smoothed, the output encoding of all the nodes becomes too similar. This problem can make important information, like the node's weight, unreadable to the actor and critic network. The most significant factor that can cause over-smoothing is the number of GNN layers, with each extra layer increasing its risk. Therefore, our five-layer Graph Encoder might need fewer layers for DTKWC. However, we also make other recommendations in our Future Research section.

6.2 Evaluation Research Questions

The previous section discussed the results of DCCA. This section indirectly answered our main and sub research questions. However, this section answers them directly. We start by doing this for the sub research questions and finish with the main research question.

Our first sub-question asked how a graph should be encoded such that DCCA can use it to compose a clique set. We used an GIN architecture as our graph encoder. This approach allowed us to implement DCCA, such that it only has to encode the whole graph once and then use the latent node encodings to compose the ideal clique set through the *encode-process-decode* paradigm. At each step, we collect the encodings of the current candidate clique set and the newfound clique using another GIN network through virtual nodes that represent the cliques and act as outputs for them. Our results showed that DCCA could outperform previous approaches in specific conditions while using this paradigm on both the runtime and the score. On larger or differently structured graphs, it performed significantly worse and even timed out on the largest graphs. However, the reason for this timeout is because DCCA uses the Pivot Bron-Kerbosch algorithm. This algorithm enumerates all the cliques in the graph; therefore, DCCA scales to the number of cliques. Other algorithms, like *EnumKOpt* (Yuan et al., 2015), optimise this

part such that not every clique has to be checked. This means that the scalability issue of DCCA, in relation to the runtime, is not caused by how we implemented DCCA but by the clique finding algorithm we used. Because of this, DCCA could use another maximal clique enumeration algorithm or other methods, which optimises the clique finding for diversity graph problems, such that it can also scale to larger graphs. Therefore, we believe that our approach of using the *encode-process-decode* paradigm by encoding the graph once and then reusing this latent encoding with another network is the right approach for DTKC and likely other diversity graph problems.

The second sub-question is related to the first question in that it asks how we should encode the structural information of a graph. To answer this, we looked at what kind of input features for the nodes we should use. We decided to use a vector of ones as the input node features, based on the paper of Abe et al. (2019) and other research (Cui et al., 2021). We showed that this method could work for DTKC. However, these node features, in combination with the network design of the graph encoder, might have hindered DCCA from learning how to compose cliques for DTKWC. The main issue we encountered in answering this sub-question was the lack of research into artificial node features - especially concerning CO problems. Therefore, we recommend future research into the topic, which will hugely benefit this research field. To summarise, we believe to have found a method to encode the structural information for DTKC but not DTKWC. However, we also think it can be improved.

Our next sub-question asked how well DCCA can generalise and scale to different graphs. Our literature review mentioned the trade-off between scalability, expressivity, and generalisation in GNN architectures. We decided to focus first on the expressivity of the network because of the novelty of DCCA. It was no surprise that DCCA could not scale and generalise to other graphs. The reason that DCCA could not scale to larger graphs is for two reasons. The first reason is the previously mentioned reason that we used the Pivot Bron-Kerbosch algorithm, which finds all the maximal cliques in the input graph. However, this explains only why DCCA would timeout on the largest graphs and only relates to the runtime. The second reason that it could not scale well to graphs with more cliques is that it likely could not fully learn the transition function of the Markov decision process (MDP) because it could not entirely observe the whole state. This problem of not being able to observe the state entirely is likely also why DCCA could not generalise well between different graphs. Therefore, we recommended reformulating the MDP as a partially observable Markov decision process (POMDP). This problem is likely the main reason, why it could not generalise between different. Therefore, we noted the second and third reasons why it could not generalise well to different graphs. We believe that the first reason is the essential explanation and that the second reason is more important than the third, but not the first. The second reason is how we generated training graphs, in that we used only the dual Barabási–Albert (BA) model. We based this approach on previous research (Abe et al., 2019; Mazyavkina et al., 2021; Cappart et al., 2021); however, this research only focused on node-level tasks and not subgraph-level tasks. Therefore, it is likely that subgraph-level tasks should use other methods or strategies to generate training graphs. Lastly, we believe the third reason why DCCA could not generalise well is that it could not learn how valuable a clique is in a given graph. We noted that two similar graphs could have varying maximal scores. This variation likely caused DCCA not to be able to learn how valuable a clique

is in relation to the graph. We do not see other NCO-RL methods having this issue (Mazyavkina et al., 2021), which can indicate that this is reason is not true or only a problem for diversity graph problems, such as DTKC and DTKWC. For this reason, we stated this as our third reason and we believe that our first and second reasons are more likely to cause this problem. Therefore, we first recommend reformulating the MDP to a partially observable Markov decision process (POMDP) and designing an algorithm based on that because this will likely help with both generalisation and scalability. If this does not improve the generalisation of this algorithm, we recommend trying other graph generator models or strategies, which we explain in our future research section. If both these fail, we recommend researching methods to normalise the reward given a graph. We will give recommendation on all three reasons in our future research section.

We also wanted to know if we could build DCCA to work for different values of k. Hence, we tested DCCA for $k = 10$, $k = 30$ and $k = 50$ and compared them the scores of *TOPKLS* for DTKC and *TOPKWCLQ* for DTKWC. Our results show that our algorithm, DCCA, works better for higher values of k when we compare DCCA to previous approaches. We also saw, that when $k = 50$, DCCA could even outperform previous approaches. We stated that two reasons probably cause this. The first is that the baseline algorithms, *TOPKLS* and *TOPKWCLQ*, perform worse for higher values of k because they can search less of the possible clique set combinations when compared to lower values k before they reach the cutoff time. We argued this by showing that the search space grows with the value of k. The second reason is that with higher values of k, DCCA gets more inputs, which primarily benefits the critic network. However, we only saw this happening at the one graph set (table 4.5) - whereas DCCA struggled with the other graph sets (tables 4.6 and 4.7).

In our last sub-research question, we asked if DCCA could work on other diversity graph problems. We tested it on DTKWC and compared them to the results of *TOPKWCLQ*. The results show that DCCA struggled learning how to compose clique sets for DTKWC and consequently likely also for other diversity graph problems. We reasoned that this is because of the over-smoothing problem, caused by to many layers in the Graph Encoder network. Nevertheless, we believe that future research can improve DCCA such that it can work for other diversity graph problems, which we clarify in our future research section.

Lastly, our main research question asked if a reinforcement learning approach can provide better results for DTKC. When we look at the score, we can conclude that DCCA does, but only on a limited amount of graphs and for higher values of k. If we compare the run time, DCCA is only faster for smaller graphs because it has to check every clique in the graph. However, DCCA is the first RL approach for any diversified graph problem, and we believe future research can improve the results further and help alleviate the problems of scalability and generalisation.

6.3 Future Research

Our results show that DCCA can learn to compose a clique set for DTKC, which scores similarly to *TOPKLS* and even outperforms *TOPKLS* for some graphs. However, DCCA can only do this when the evaluation graphs are generated by the dual Barabási–Albert

model with the same parameters and, therefore, cannot generalise and scale well to different and larger graphs. Another issue is its scalability to larger graphs because it has to check every clique in the graph. We found that the scalability problem of the runtime was not caused by DCCA, but by the use of Pivot Bron-Kerbosch algorithm. Therefore, we start by making recommendations to solve this problem.

The simplest solution to solve the scalability problem for DTKC is to combine DCCA with *EnumKOpt* (Yuan et al., 2015). Utilising *EnumKOpt* will limit the number of cliques that DCCA has to check because *EnumKOpt* prunes non-viable solutions. DCCA will handle the composition of the clique set. However, this combination would only work for DTKC and not DTKWC or other diversify graph problems.

Another solution is to create another agent who finds the cliques for DCCA. This setup would result in a cooperative multi-agent environment where DCCA still functions the same, but the other agent finds the cliques based on the current candidate clique set (Oroojlooyjadid and Hajinezhad, 2019). Kim et al. (2021) implemented a similar setup for the travelling salesman problem (TSP), in which one policy tried to find a solution and another policy tried to improve the found solution. In such an approach, both agents might be able to share the graph encoder by using the same latent space $\mathcal{Z}_{\mathcal{G}}$ for the clique finding and the clique comparison. We recommend comparing such an approach to when both agents do not share the graph encoder. However, this agent will likely face the same problem as DCCA, in that it will not know which cliques are already seen.

In discussing the results, we argued that one of DCCA main issues is the inability to learn to predict which cliques are already enumerated and which cliques still need to be enumerated, which is more noticeable for larger graphs and graphs that are structurally different to the generated graphs. DCCA likely cannot completely observe the current state, and, therefore, the problem should be formulated as a POMDP (Åström, Karl Johan, 1965). However, we think that including an RNN would improve the results of DCCA significantly (Kapturowski et al., 2019). The main argument behind this hypothesis is that by using an RNN network, DCCA can learn which cliques are already enumerated and which are not. The input of the RNN could be the coverage of the clique set with the newfound clique $\text{Cov}(\mathcal{D} \cup \{C_T\})$. The output of the RNN could be combined with the input for both the actor and critic networks. We see that other NCO-RL algorithms also use RNN architectures, mostly for TSP, to encode the state space (Mazyavkina et al., 2021).

The discussion of the results showed that DCCA could not generalise well between differently structured graphs. We argued, as our second reason, that this is because of how generated training graphs. We based our approach for this on previous research (Abe et al., 2019; Mazyavkina et al., 2021; Cappart et al., 2021). This research showed that algorithms trained on generated graphs could generalise well to different graphs; however, all of this research was focused on node-level tasks and, again, not subgraph-level tasks. Therefore, we believe that using only the dual BA model was not the right approach for our thesis. Future research could use other graph generation models, such as those used in community detection research. Examples of these are the Stochastic Block Model (Holland et al., 1983) or LFR Benchmark algorithm (Lancichinetti et al., 2008). Other options would be to train future improvements of DCCA on a combination of different graph generator models. Lastly, we believe another option might be to use

methods like NetGAN (Bojchevski et al., 2018). NetGAN can generate graphs similar to real-world graphs and is based on recent breakthroughs with Generative Adversarial Networks and their ability to generate images. One of these proposed approaches will likely improve DCCA's ability to generalise between different graphs; however, we cannot state which is the best candidate. Therefore, we recommend that future research compares those methods. This comparison will benefit DTKC, other diversity graph problems and likely other research that needs to generate graphs to train a GNN for subgraph-level tasks.

In discussion of the results, we stated, as our third reason, that one of the issues of DCCA is that the maximum score of graph can vary wildly between graphs, even though those graphs can have the same size. Our reward function does not normalise the reward, which meant it would be harder for DCCA to learn how valuable a clique is, given that graph. We stated in section 3.1.1 that either the maximum clique for DTKC or the maximum weighted clique for DTKWC might be used for normalisation. Nevertheless, this would likely not be beneficial because it limits any future approach to only DTKC and DTKWC. Moreover, there is no evidence that either the maximum clique or the maximum weighted clique hold any relation to the maximum score. A solution for this would be to find another method or value of normalising the reward, by domain-knowledge of the problem. However, this will likely be infeasible due to DTKC and other diversity graph problems being NP-Hard, and thus, finding or approximating the maximum reward by domain-knowledge almost impossible. Another reason we do not recommend this approach is that each diversity graph problem likely has a different method or value for the normalisation, while the intent is to have DCCA easily be extendable to other diversity graph problems. Therefore, we believe that another solution might be to use *adaptive normalisation*, such as POP-ART (van Hasselt et al., 2016). With POP-ART, the agent learns how to scale the reward, which might allow DCCA to learn how to scale the reward based on the graph. Nevertheless, we want to note that agents trained with POP-ART performed significantly worse in some tests, according to their results. Therefore, it might be necessary to research a similar methods to POP-ART but that is focused on diversity graph problems.

In our discussion of the runtime, we stated that the usage of the Pivot Bron-Kerbosch algorithm affected the total run time of DCCA. Due to the scope of this thesis, we did not have the resources to optimise this; therefore, we believe improving this process can significantly improve the runtime. Options for this are to rewrite DCCA in C++ or use another algorithm for finding the cliques, which we described earlier in this section. However, finding or designing a different maximal clique enumeration algorithm would need a focused research approach. Therefore, a simple adjustment would be to use a more optimised version of the Bron-Kerbosch algorithm, such as the algorithm of Segundo et al. (2018). We also stated that higher values of k increased the run time. One shortcoming of DCCA is that it rechecks every clique at each step of the episode, which should not be necessary because the output for each clique is independent of the other cliques. Therefore, we believe this process should be optimised such that it checks the clique once and then keeps the result in memory until the clique is removed from the clique set.

Our results showed that DCCA could not learn how to compose a clique set for DTKWC. We argued that the most probable reason for this is a combination of the

input features and the network design of the graph encoder, which caused the node's weights to be unreadable. However, we still believe DCCA could also function for DTKWC by adjusting both. The capability of the graph encoder could be enhanced by using Jumping Knowledge (Xu et al., 2018b). Jumping Knowledge combines the output of the multiple layers of a GNN as the final output of that GNN. Hence, it should alleviate the over-smoothing problem, which caused the node's weight to be unreadable for DCCA because of five GIN layers. To test if Jumping Knowledge will improve results, one could first test the GNN network by trying to let it learn the combined weights of cliques in a supervised manner.

We also believe that the results of DCCA could be improved by adjusting the input features. We see that the research into adding node features is limited, especially for graph CO problems (Cappart et al., 2021). Therefore, we strongly recommend researching which node features should be added for graph CO problems. This research could then act as starting point for research into neural combinatorial optimisation (NCO) to decide which features to add to which problem.

Aside from the node features, we also noticed the lack of research into subgraph-level task GNN architectures or embedding methods. The only considerable research we found into the topic was SubGNN (Alsentzer et al., 2020). This made our research significantly harder because we had to implement our solution without any meaningful examples of subgraph-level approaches. Therefore, we believe that research into either comparing current methods for subgraph-level tasks or finding new methods of encoding subgraphs will benefit DCCA and other future subgraph-level task approaches.

If future research shows that future approaches can compose a clique set for DTKWC. In that case, these approaches should be tested on other diversity graph problems, such as the diversity top-k s-plex problem (Wu and Yin, 2021a) and diversified top-k subgraph querying (Fan et al., 2013). After that, these approaches could be extended to function on max k-cover problems. However, to do this would likely require a complete overhaul of the network architecture.

We implemented DCCA using PPO, an actor-critic policy gradient method and saw its benefits for DTKC. However, PPO and other policy gradient algorithm might not be the best RL algorithm for DTKC and other diversity graph problems. We, therefore, recommend also trying to implement DCCA using DQN (Mnih et al., 2013). One of the main limitations of DQN is that it is only compatible with a discrete action space. However, this limitation is irrelevant because DTKC and any other diversity graph will likely always have a discrete action space. Hence, we believe that DQN and its extended version, such as Rainbow DQN (Hessel et al., 2018), could improve the performance of future approaches at the cost of increased training time.

Besides DQN, we believe that Neural MCTS might even result in better performance than either DQN or PPO. This believe mainly stem from other algorithms that use neural MCTS. For example, we see neural MCTS being implemented for all kinds of optimisation problems, ranging from fluid-structure topology optimisation (Gaymann and Montomoli, 2019) to the bin packing problem (Laterre et al., 2018). Moreover, two examples stand out for us regarding Neural MCTS. The first is the algorithm by Abe et al. (2019), on which we based our graph encoder network. One of the problems it worked for was the maximum clique, which is highly related to DTKC, and showed its potential to graph problems in combination with GNN. The other example is the algo-

rithm of Zou et al. (2019), which tried to find the top k diverse recommendations from a database. This algorithm showed the potential of neural MCTS to find a diverse top-k set.

A lot of research states the promise of neural combinatorial optimisation (NCO) to function on instances of CO problems that previous algorithms could not because of the need to abstract the input data or because of non-linear relations in the data. Nevertheless, we do not see any benchmark problems that explicitly test these promised properties. We already see the promise of this field with the paper of Mirhoseini et al. (2021), which introduced an RL algorithm that can learn how to design efficient computer chips by formulating it as a CO problem. This problem could act as a benchmark, but we also recommend adjusting existing CO problems, such as the travelling salesman problem (TSP) and DTKC, to have natural inputs or non-linear relations in the data. These benchmarks should significantly advance this research field because they will allow better research into solutions that can be applied to the real world and to cases for which classical CO algorithms cannot be adapted.

6.4 Conclusion

This thesis examined how to use reinforcement learning for the diversified top-k clique search problem (DTKC). Our research question asked whether a reinforcement learning approach will improve the results of DTKC on either the runtime or the score. Deep Clique Comparison Agent (DCCA), outperformed *TOPKLS*, a previous approach, on smaller generated graphs when $k = 50$ but performed significantly worse on larger generated graphs and real-world graphs and did not work at all for the diversified top-k weighted clique search problem (DTKWC). Nevertheless, these insights are important because this is the first time reinforcement learning and deep learning methods are used for DTKC or any other diversity graph problem. We also presume that DTKC is one of the most complex combinatorial optimisation problems to which a deep RL approach is tried for. We showed that a deep RL algorithm could compose a diverse clique set through the *encode-process-decode* paradigm. Showing that this paradigm can work in such an environment might be our most significant contribution to the fields of deep learning and combinatorial optimisation. It could lessen the problem of scalability that most deep learning methods currently have in this research field.

In our discussion, we made recommendations for future research, that may help a future approaches with other diversity graph problems, such as DTKC, and what is necessary to improve the results with DTKC. We believe that deep RL is the right approach for DTKC and similar problems, especially for higher values of k. Lastly, we expect deep RL algorithms, such as DCCA, to be more useful in domains where the data consists of natural inputs and can not be abstractified for classical combinatorial optimisation algorithms.

Glossary

BA Barabási–Albert. 2, 14, 15, 28, 31, 41, 42, 44, 50, 54, 64, 67, 77, 82, 83, 113, 114, 117, 119, 122, 123, 126, 128, 131, 132, 135, 137, 140, 141, 144, 146

CO combinatorial optimisation. 4, 5, 7, 9, 14–16, 20, 27–31, 80, 82, 86, 87

DCCA Deep Clique Comparison Agent. 1, 2, 5, 13, 14, 31, 33, 34, 36–41, 45–54, 56–61, 63–65, 67–69, 71–74, 76, 80–87, 112, 121

DQN deep Q-Learning. 22, 23, 39, 86

DTKC diversified top-k clique search problem. 1, 5–8, 12–16, 18, 20, 27–33, 39–41, 46, 47, 49, 50, 64, 77, 78, 80–87, 112, 130

DTKSP diversified top-k s-plex search problem. 8

DTKSQ diversified top-k subgraph querying. 8

DTKWC diversified top-k weighted clique search problem. 1, 8, 11, 13, 14, 18, 27, 31–33, 39, 41, 46, 47, 49, 64, 67, 77, 78, 80–87, 112, 131

ECC enhanced configuration checking. 20, 21

ER Erdős-Rényi. 14, 15, 28, 41

GAE Generalised Advantage Estimation. 24, 46, 47

GAT Graph Attention Networks. 26, 35

GCN Graph Convolutional Networks. 26, 30, 34

GIN Graph Isomorphic Networks. 1, 14, 26, 27, 29, 34, 35, 40, 46, 80, 81, 86

GNN Graph Neural Networks. 11, 12, 14, 26, 27, 29, 39, 40, 82, 86

MC maximum clique problem. 18, 29

MCE maximal clique enumeration. 7, 16, 19, 82, 85

MCTS Monte Carlo tree search. 9, 22, 26, 29, 30, 39, 86, 87

MDP Markov decision process. 9, 31, 40, 82, 83

MIS maximum independent set problem. 29

MLP multilayer perceptron. 11, 27, 34, 40, 46

NCO neural combinatorial optimisation. 28, 29, 86, 87

NCO-RL neural combinatorial optimisation with reinforcement learning. 28–30, 40, 78, 79, 83, 84

POMDP partially observable Markov decision process. 78, 79, 82–84

PPO Proximal Policy Optimization Algorithms. 1, 14, 22, 24, 25, 30, 33, 36, 39, 40, 86

RL reinforcement learning. 4, 5, 8–10, 13–15, 22, 23, 26, 28–31, 33, 34, 39, 40, 83, 86, 87

TD temporal difference. 10, 23, 24

TRPO Trust Region Proximal Optimization. 24, 25

TSP travelling salesman problem. 4, 5, 15, 16, 29, 84, 87

Bibliography

Aarts, E. and Lenstra, J., editors (1997). *Local search in combinatorial optimization*. Wiley-Interscience series in discrete mathematics and optimization. Wiley-Interscience.

Abe, K., Xu, Z., Sato, I., and Sugiyama, M. (2019). Solving np-hard problems on graphs by reinforcement learning without domain knowledge. *CoRR*, abs/1905.11623.

Abramé, A., Habet, D., and Toumi, D. (2016). Improving configuration checking for satisfiable random k-SAT instances. *Annals of Mathematics and Artificial Intelligence*, 79(1-3):5–24.

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375.

Ahn, S., Seo, Y., and Shin, J. (2020). Learning what to defer for maximum independent sets. *CoRR*, abs/2006.09607.

Akanmu, S., Garg, R., and Gilal, A. (2019). Towards an improved strategy for solving multi- armed bandit problem. *International Journal of Innovative Technology and Exploring Engineering*, 10:5060–5064.

Albert, R. and Barabási, A.-L. (2000). Topology of evolving networks: Local events and universality. *Physical Review Letters*, 85(24):5234–5237.

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97.

Alguliyev, R., Aliguliyev, R., and Yusifov, F. (2021). Graph modelling for tracking the COVID-19 pandemic spread. *Infectious Disease Modelling*, 6:112–122.

Alsentzer, E., Finlayson, S., Li, M., and Zitnik, M. (2020). Subgraph neural networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 8017–8029. Curran Associates, Inc.

Applegate, D., Bixby, R., Chvátal, V., Cook, W., and Helsgaun, K. (2010). Optimal tour of sweden. The Traveling Salesman Problem.

Baek, M., DiMaio, F., Anishchenko, I., Dauparas, J., Ovchinnikov, S., Lee, G. R., Wang, J., Cong, Q., Kinch, L. N., Schaeffer, R. D., Millán, C., Park, H., Adams, C., Glassman, C. R., DeGiovanni, A., Pereira, J. H., Rodrigues, A. V., van Dijk, A. A., Ebrecht, A. C., Opperman, D. J., Sagmeister, T., Buhlheller, C., Pavkov-Keller, T., Rathinaswamy, M. K., Dalwadi, U., Yip, C. K., Burke, J. E., Garcia, K. C., Grishin, N. V., Adams, P. D., Read, R. J., and Baker, D. (2021). Accurate prediction of protein structures and interactions using a 3-track network. *bioRxiv*.

Balasundaram, B., Butenko, S., and Hicks, I. V. (2011). Clique relaxations in social network analysis: The maximumk-plex problem. *Operations Research*, 59(1):133–142.

Bellman, R. (1957). A markovian decision process. *Indiana University Mathematics Journal*, 6(4):679–684.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940.

Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. (2018). NetGAN: Generating graphs via random walks. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 610–619. PMLR.

Boppana, R. and Halldórsson, M. M. (1992). Approximating maximum independent sets by excluding subgraphs. *BIT*, 32(2):180–196.

Bron, C. and Kerbosch, J. (1973). Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577.

Cai, S., Jie, Z., and Su, K. (2015). An effective variable selection heuristic in SLS for weighted max-2-SAT. *Journal of Heuristics*, 21(3):433–456.

Cai, S. and Su, K. (2013). Local search for boolean satisfiability with configuration checking and subscore. *Artificial Intelligence*, 204:75–98.

Cai, S., Su, K., and Sattar, A. (2011). Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9):1672–1696.

Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Velickovic, P. (2021). Combinatorial optimization and reasoning with graph neural networks. *CoRR*, abs/2102.09544.

Cappart, Q., Goutierre, E., Bergman, D., and Rousseau, L. (2018). Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. *CoRR*, abs/1809.03359.

Cappart, Q., Moisan, T., Rousseau, L., Prémont-Schwarz, I., and Ciré, A. A. (2020). Combining reinforcement learning and constraint programming for combinatorial optimization. *CoRR*, abs/2006.01610.

Cazals, F. and Karande, C. (2008). A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407(1):564–568.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. (2019). Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *CoRR*, abs/1909.03211.

Chen, X. and Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E. A., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 6278–6289.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. ACM Press.

Coulom, R. (2007). Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games*, pages 72–83. Springer Berlin Heidelberg.

Croce, F. D. and Paschos, V. T. (2012). Efficient algorithms for the max k -vertex cover problem. In *Lecture Notes in Computer Science*, pages 295–309. Springer Berlin Heidelberg.

Cui, H., Lu, Z., Li, P., and Yang, C. (2021). On positional and structural node features for graph neural networks on non-attributed graphs. *CoRR*, abs/2107.01495.

Dai, H., Dai, B., and Song, L. (2016). Discriminative embeddings of latent variable models for structured data. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 2702–2711. JMLR.org.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., and Rousseau, L.-M. (2018). Learning heuristics for the tsp by policy gradient. In van Hoeve, W.-J., editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 170–181, Cham. Springer International Publishing.

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. *CoRR*, abs/1509.09292.

Eppstein, D., Löffler, M., and Strash, D. (2010). Listing all maximal cliques in sparse graphs in near-optimal time. *CoRR*, abs/1006.5440.

Erdös, P. and Rényi, A. (1959). On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290.

Errica, F., Podda, M., Bacciu, D., and Micheli, A. (2019). A fair comparison of graph neural networks for graph classification. *CoRR*, abs/1912.09893.

Euler, L. (1741). Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140.

Fan, W., Wang, X., and Wu, Y. (2013). Diversified top-k graph pattern matching. *Proc. VLDB Endow.*, 6(13):1510–1521.

Feige, U. (1998). A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric. cite arxiv:1903.02428.

Figueiredo, D. R., Ribeiro, L. F. R., and Saverese, P. H. P. (2017). struc2vec: Learning node representations from structural identity. *CoRR*, abs/1704.03165.

Gaymann, A. and Montomoli, F. (2019). Deep neural network and monte carlo tree search applied to fluid-structure topology optimization. *Scientific Reports*, 9(1).

Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA. Curran Associates Inc.

Hamrick, J. B., Allen, K. R., Bapst, V., Zhu, T., McKee, K. R., Tenenbaum, J. B., and Battaglia, P. W. (2018). Relational inductive bias for physical construction in humans and machines. *CoRR*, abs/1806.01203.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Holland, P. W., Laskey, K. B., and Leinhardt, S. (1983). Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137.

Holme, P. and Kim, B. J. (2002). Growing scale-free networks with tunable clustering. *Phys. Rev. E*, 65:026107.

Huber, W., Carey, V. J., Long, L., Falcon, S., and Gentleman, R. (2007). Graphs in molecular biology. *BMC Bioinformatics*, 8(S6).

Kapturowski, S., Ostrovski, G., Dabney, W., Quan, J., and Munos, R. (2019). Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer US.

Kim, M., Park, J., and kim, j. (2021). Learning collaborative policies to solve np-hard routing problems. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 10418–10430. Curran Associates, Inc.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., and Perez, P. (2021). Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–18.

Konda, V. R. and Tsitsiklis, J. N. (2003). OnActor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166.

Kool, W., van Hoof, H., and Welling, M. (2019). Attention, learn to solve routing problems! In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Kostrikov, I. (2018). Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

Laarhoven, P. J. M. and Aarts, E. H. L. (1987). *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, USA.

Lancichinetti, A., Fortunato, S., and Radicchi, F. (2008). Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4).

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247.

Laterre, A., Fu, Y., Jabri, M. K., Cohen, A.-S., Kas, D., Hajjar, K., Dahl, T. S., Kerkeni, A., and Beguir, K. (2018). Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization. *ArXiv*, abs/1807.01672.

Li, R., Hu, S., Zhang, H., and Yin, M. (2016). An efficient local search framework for the minimum weighted vertex cover problem. *Information Sciences*, 372:428–445.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated graph sequence neural networks. cite arxiv:1511.05493Comment: Published as a conference paper in ICLR 2016. Fixed a typo.

Liang, Y. (1996). Combinatorial optimization by hopfield networks using adjusting neurons. *Information Sciences*, 94(1-4):261–276.

Lichtenstein, D. and Sipser, M. (1980). GO is polynomial-space hard. *Journal of the ACM*, 27(2):393–401.

Lin, X., Yuan, Y., Zhang, Q., and Zhang, Y. (2007). Selecting stars: The k most representative skyline operator. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE.

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400.

Mańdziuk, J. (1996). Solving the travelling salesman problem with a hopfield-type neural network. *Demonstratio Mathematica*, 29:219–231.

Mignon, A. and A. Rocha, R. L. (2017). An adaptive implementation of ϵ-greedy in reinforcement learning. *Procedia Computer Science*, 109:1146–1151.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Le, Q. V., Laudon, J., Ho, R., Carpenter, R., and Dean, J. (2021). A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Moon, J. W. and Moser, L. (1965). On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28.

Moshiri, N. (2018). The dual-barabási-albert model.

Oroojlooyjadid, A. and Hajinezhad, D. (2019). A review of cooperative multi-agent deep reinforcement learning. *CoRR*, abs/1908.03963.

Otte, E. and Rousseau, R. (2002). Social network analysis: a powerful strategy, also for the information sciences. *Journal of Information Science*, 28(6):441–453.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab. Previous number = SIDL-WP-1999-0120.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710, New York, NY, USA. ACM.

Rossi, R. A. and Ahmed, N. K. (2015). The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England.

Ryan, C. (2003). Evolutionary algorithms and metaheuristics. In Meyers, R. A., editor, *Encyclopedia of Physical Science and Technology (Third Edition)*, pages 673–685. Academic Press, New York, third edition edition.

Sanchez-Lengeling, B., Reif, E., Pearce, A., and Wiltschko, A. B. (2021). A gentle introduction to graph neural networks. *Distill*. https://distill.pub/2021/gnn-intro.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. P., and Silver, D. (2019). Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015a). Trust region policy optimization. *CoRR*, abs/1502.05477.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Segundo, P. S., Artieda, J., and Strash, D. (2018). Efficiently enumerating all maximal cliques with bit-parallelism. *Comput. Oper. Res.*, 92:37–46.

Seidman, S. B. and Foster, B. L. (1978). A graph-theoretic generalization of the clique concept. *The Journal of Mathematical Sociology*, 6(1):139–154.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423.

Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T. P., Simonyan, K., and Hassabis, D. (2017a). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Driessche, G., Graepel, T., and Hassabis, D. (2017b). Mastering the game of go without human knowledge. *Nature*, 550:354–359.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Tomita, E., Tanaka, A., and Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42. Computing and Combinatorics.

Tunyasuvunakool, K., Adler, J., Wu, Z., Green, T., Zielinski, M., Žídek, A., Bridgland, A., Cowie, A., Meyer, C., Laydon, A., Velankar, S., Kleywegt, G., Bateman, A., Evans, R., Pritzel, A., Figurnov, M., Ronneberger, O., Bates, R., Kohl, S., and Hassabis, D. (2021). Highly accurate protein structure prediction for the human proteome. *Nature*, 596:1–9.

van Hasselt, H. P., Guez, A., Guez, A., Hessel, M., Mnih, V., and Silver, D. (2016). Learning values across many orders of magnitude. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2017). Graph Attention Networks. *arXiv e-prints*, page arXiv:1710.10903.

Villanueva, J. C. (2018). How many atoms are there in the universe?

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2692–2700, Cambridge, MA, USA. MIT Press.

Wang, J., Cheng, J., and Fu, A. W.-C. (2013). Redundancy-aware maximal cliques. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM.

Wang, X. and Zhan, H. (2018). Approximating diversified top-k graph pattern matching. In Hartmann, S., Ma, H., Hameurlain, A., Pernul, G., and Wagner, R. R., editors, *Database and Expert Systems Applications*, pages 407–423, Cham. Springer International Publishing.

Wang, Y., Liang, S., Yang, Q., and Wang, X. (2016). Facile synthesis of dendritic cu by electroless reaction of cu-al alloys in multiphase solution. *Applied Surface Science*, 387:805–811.

Warren, J. S. and Hicks, I. V. (2006). Combinatorial branch-and-bound for the maximum weight independent set problem.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4):279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer US.

Wu, J., Li, C.-M., Jiang, L., Zhou, J., and Yin, M. (2020). Local search for diversified top-k clique search problem. *Computers & Operations Research*, 116:104867.

Wu, J. and Yin, M. (2021a). Local search for diversified top-k s-plex search problem (student abstract). *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(18):15929–15930.

Wu, J. and Yin, M. (2021b). A restart local search for solving diversified top-k weight clique search problem. *Mathematics*, 9(21).

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2019). A comprehensive survey on graph neural networks. cite arxiv:1901.00596Comment: updated tables and references.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018a). How powerful are graph neural networks? *CoRR*, abs/1810.00826.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., and Jegelka, S. (2018b). Representation learning on graphs with jumping knowledge networks. *CoRR*, abs/1806.03536.

Yang, Z., Fu, A. W.-C., and Liu, R. (2016). Diversified top-k subgraph querying in a large graph. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 1167–1182, New York, NY, USA. Association for Computing Machinery.

Yuan, L., Qin, L., Lin, X., Chang, L., and Zhang, W. (2015). Diversified top-k clique search. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE.

Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P. S., and Chi, X. (2020). Learning to dispatch for job shop scheduling via deep reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1621–1632. Curran Associates, Inc.

Zhang, S. and Sutton, R. S. (2017). A deeper look at experience replay. *CoRR*, abs/1712.01275.

Zheng, J., He, K., Zhou, J., Jin, Y., and Li, C. (2020). Combining reinforcement learning with lin-kernighan-helsgaun algorithm for the traveling salesman problem. *CoRR*, abs/2012.04461.

Zou, L., Xia, L., Ding, Z., Yin, D., Song, J., and Liu, W. (2019). Reinforcement learning to diversify top-n recommendation. In Li, G., Yang, J., Gama, J., Natwichai, J., and Tong, Y., editors, *Database Systems for Advanced Applications*, pages 104–120, Cham. Springer International Publishing.

Åström, Karl Johan (1965). Optimal Control of Markov Processes with Incomplete State Information I. 10:174–205.

Appendix A

Graph Analysis

A.1 The Barabási–Albert model and the Erdős-Rényi model

A.1.1 The Barabási–Albert model

Value	Mean	SD	Max	Min
Mean Clique Size	3.002	0.024	3.12	2.933
SD Clique Size	1.015	0.034	1.148	0.901
Max Clique Size	9.636	0.744	12	8
Number of Cliques	19567.706	134.208	19998	19143
$\|V\|$	1500	0	1500	1500
$\|E\|$	22275	0	22275	22275
Mean Degree	29.7	0	29.7	29.7

Table A.1: Generated with BA-model with $n = 1500$ and $m = 15$

Value	Mean	SD	Max	Min
Mean Clique Size	4.532	0.068	4.789	4.337
SD Clique Size	1.964	0.08	2.222	1.685
Max Clique Size	15.372	0.971	19	13
Number of Cliques	68180.907	1293.997	73582	64289
$\|V\|$	1500	0	1500	1500
$\|E\|$	44100	0	44100	44100
Mean Degree	58.8	0	58.8	58.8

Table A.2: Generated with BA-model with $n = 1500$ and $m = 20$

A.1.2 The Erdős-Rényi model

Value	Mean	SD	Max	Min
Mean Clique Size	2.055	0.003	2.064	2.047
SD Clique Size	0.228	0.006	0.245	0.211
Max Clique Size	3.195	0.396	4	3
Number of Cliques	10240.125	89.244	10557	9955
$\|V\|$	1500	0	1500	1500
$\|E\|$	11243.789	107.53	11577	10877
Mean Degree	14.992	0.143	15.436	14.503

Table A.3: Generated with ER-model with $n = 1500$ and $p = 0.01$

Value	Mean	SD	Max	Min
Mean Clique Size	2.265	0.006	2.285	2.247
SD Clique Size	0.443	0.003	0.454	0.433
Max Clique Size	4.001	0.032	5	4
Number of Cliques	16800.388	94.103	17055	16465
$\|V\|$	1500	0	1500	1500
$\|E\|$	22482.919	151.075	22909	22051
Mean Degree	29.977	0.201	30.545	29.401

Table A.4: Generated with ER-model with $n = 1500$ and $p = 0.02$

Appendix B

Training Statistics

B.1 Explained Variance

The figures in this section of the appendix show the explained variance during the run. The explained variance indicates how much of the variance is accounted for in predicting the value of the state. The most optimal value for the explained variance is 1, which indicates that the value network explains all the variance. Lower values for the explained variance indicate issues with the value network.

B.1.1 Diversified Top-k Clique Search

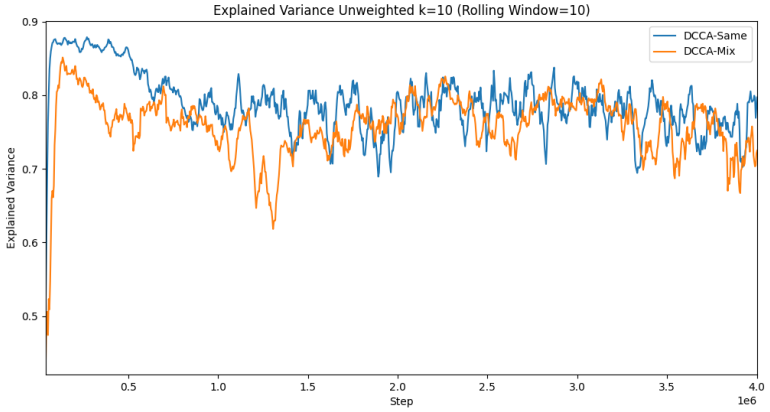


Figure B.1: The explained variance for $k = 10$

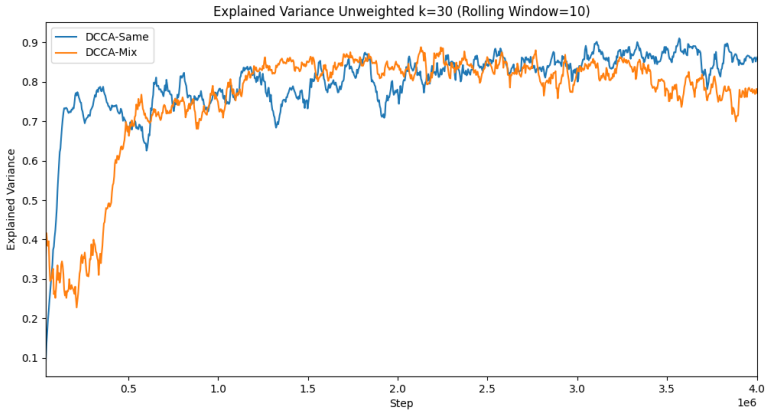


Figure B.2: The explained variance for $k = 30$

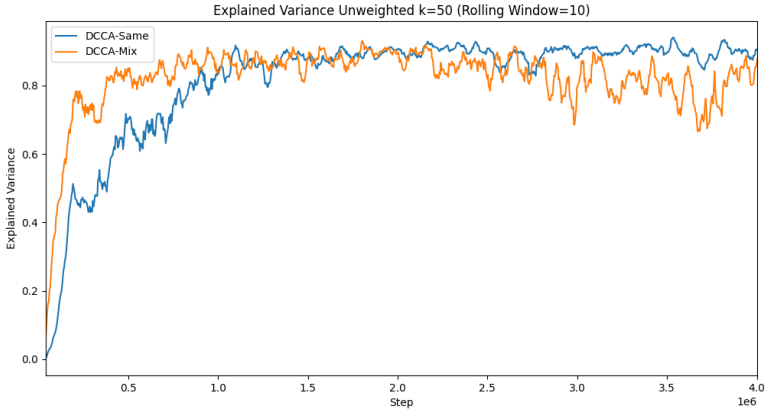


Figure B.3: The explained variance for $k = 50$

B.1.2 Diversified Top-k Weighted Clique Search

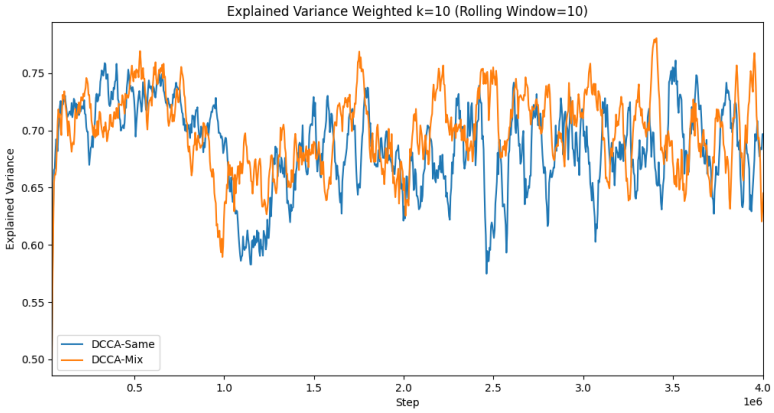


Figure B.4: The explained variance for $k = 10$

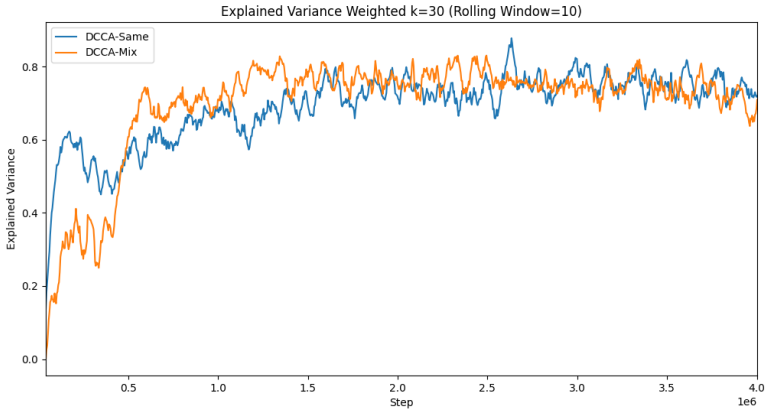


Figure B.5: The explained variance for $k = 30$

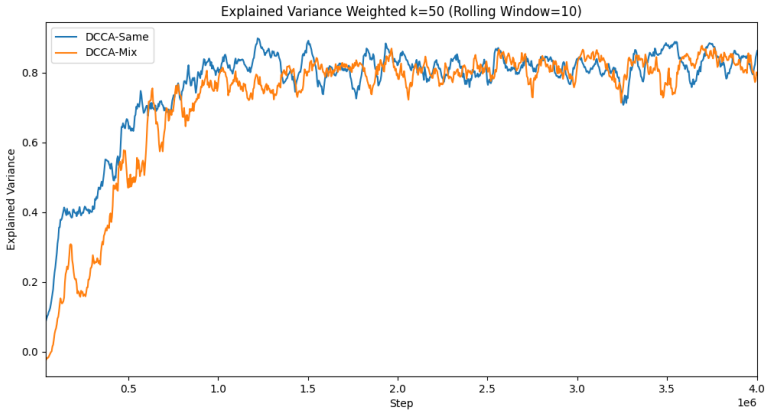


Figure B.6: The explained variance for $k = 50$

B.2 Reward Sum

The figures of the reward sum show the summation of all the rewards for that episode. Therefore, a higher value is always strictly better.

B.2.1 Diversified Top-k Clique Search

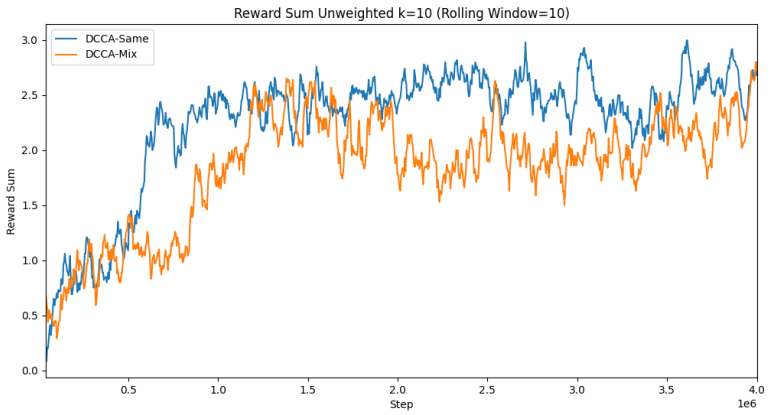


Figure B.7: The reward sum for $k = 10$

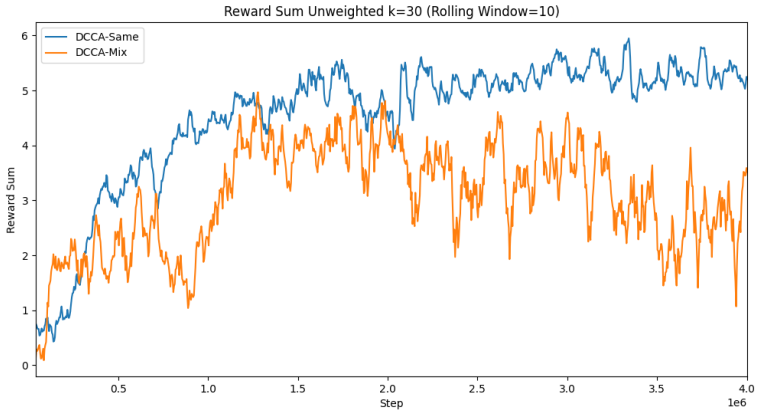


Figure B.8: The reward sum for $k = 30$

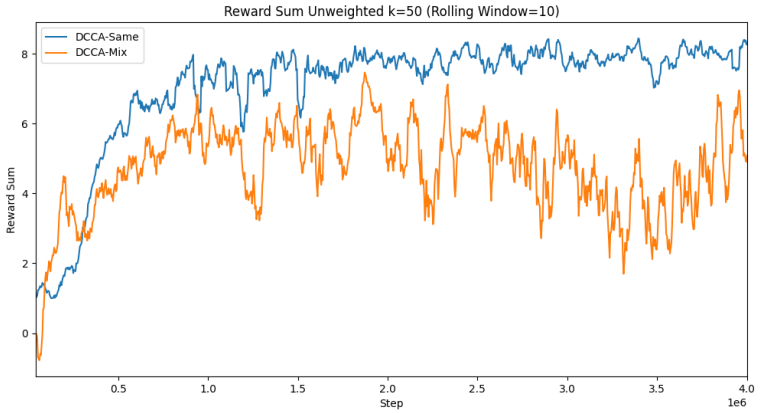


Figure B.9: The reward sum for $k = 50$

B.2.2 Diversified Top-k Weighted Clique Search

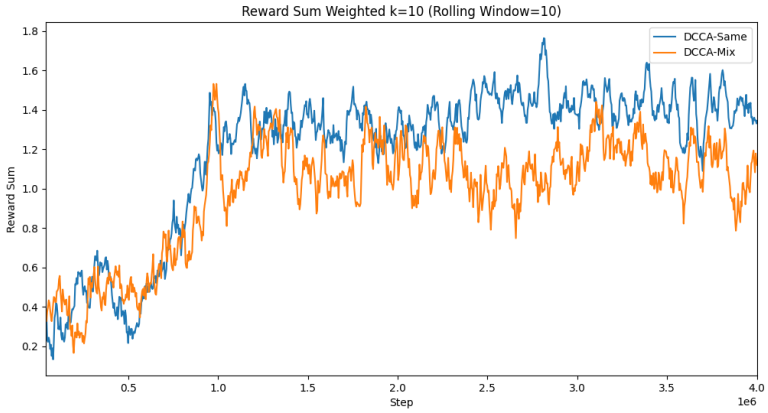


Figure B.10: The reward sum for $k = 10$

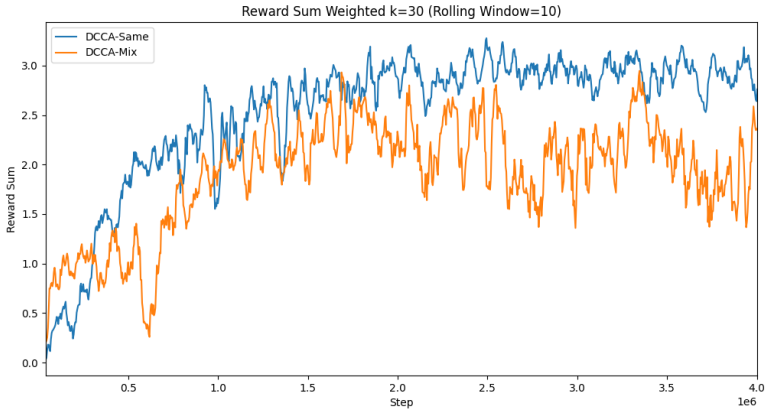


Figure B.11: The reward sum for $k = 30$

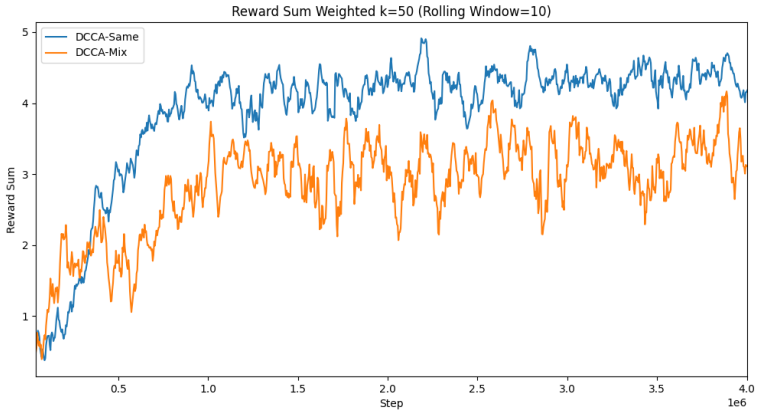


Figure B.12: The reward sum for $k = 50$

B.3 Distribution Entropy

The distribution entropy is the Shannon Entropy (Shannon, 1948) over all the actions during training. At the start of training, it will be maximal because each action has an equal probability of being selected. During training, the entropy should gradually decrease because DCCA becomes more certain about which action is the best given that state.

B.3.1 Diversified Top-k Clique Search

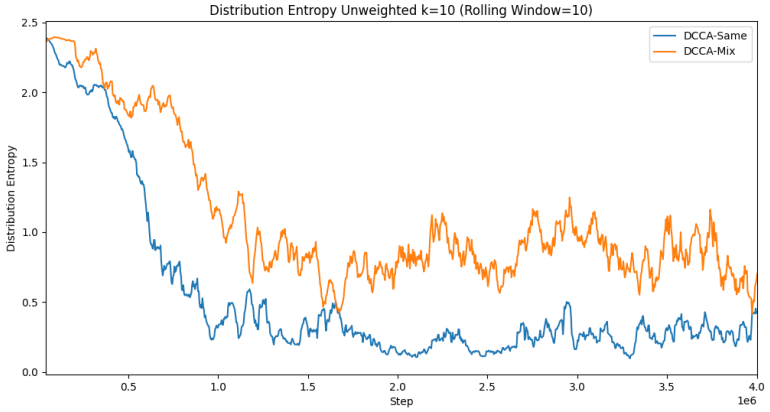


Figure B.13: The distribution entropy for $k = 10$

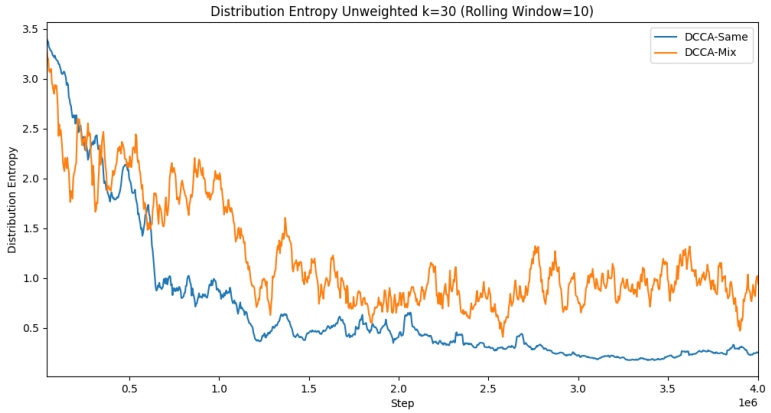


Figure B.14: The distribution entropy for $k = 30$

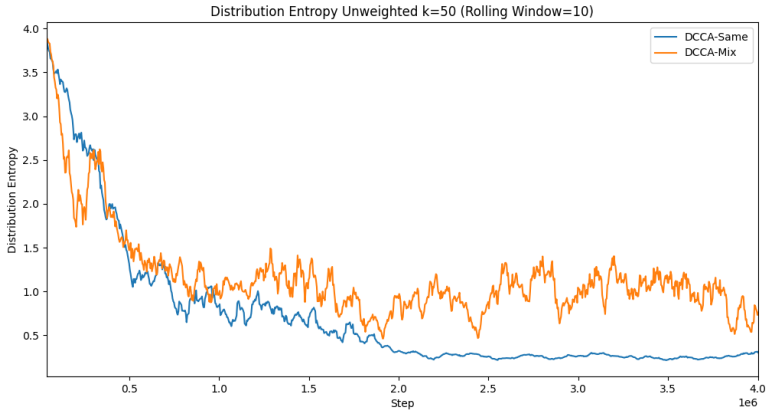


Figure B.15: The distribution entropy for $k = 50$

B.3.2 Diversified Top-k Weighted Clique Search

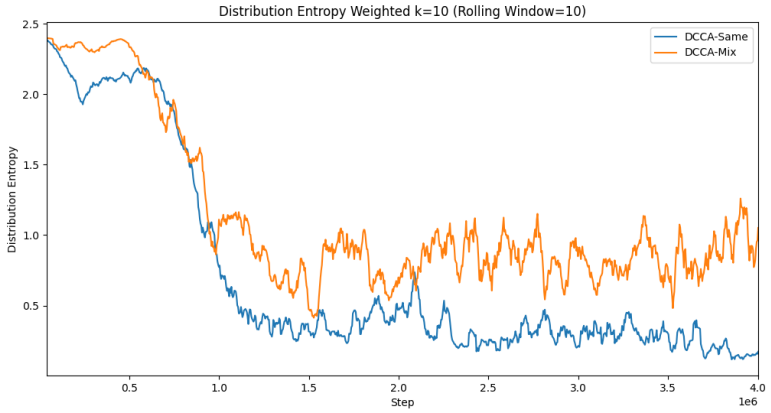


Figure B.16: The distribution entropy for $k = 10$

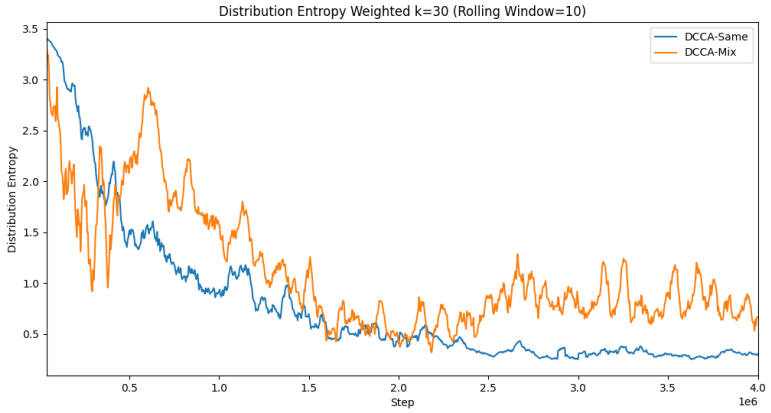


Figure B.17: The distribution entropy for $k = 30$

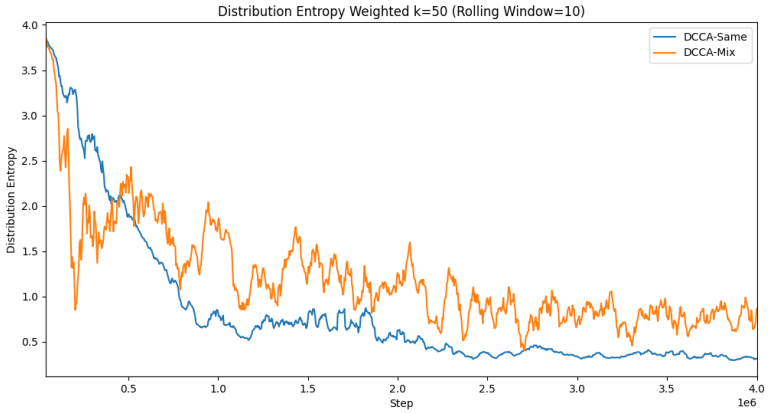


Figure B.18: The distribution entropy for $k = 50$

Appendix C

Results

In this appendix, we show the full results of our experiments. Each compared setup has its' section; therefore, we have three sections in this appendix, the results of DCCA-Same, the results of DCCA-Mix, and the results of *TOPKLS* or *TOPKWCLQ*. These sections are then separated into subsections for each problem: the diversified top-k clique search problem (DTKC) and diversified top-k weighted clique search problem (DTKWC). In each of these subsections, we organise the results by one which evaluation graph set it was tested.

C.1 DCCA-Same

C.1.1 Diversified top-k clique search problem

This subsection shows the results of DCCA-Same. We start by explaining what each result is. In each table, we first note the end and max scores. The end score is the score, which is either the coverage or the total of the coverage of the returned clique set. The max score is the highest found score during the run. The following column shows the percentual difference between the max and end scores, with 0% meaning that the end and max scores are the same. After that, we show the total runtime and the time at which the max score was found. Lastly, we show the total number of cliques checked for that graph and at which step the max score was found.

Dual Barabási–Albert model - Same Parameters

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	49	50	-2.0%	6.8	2.2	8380	2695
graph_1	41	45	-8.89%	6.85	0.83	8414	1019
graph_2	44	44	0.0%	6.2	1.48	7761	1841
graph_3	46	47	-2.13%	6.23	1.44	7649	1734
graph_4	42	48	-12.5%	6.56	1.16	8035	1414
graph_5	48	50	-4.0%	6.5	1.42	7987	1557
graph_6	45	45	0.0%	6.41	4.78	7938	5964
graph_7	42	42	0.0%	6.45	0.56	7747	653
graph_8	47	47	0.0%	6.21	3.3	7519	3935
graph_9	39	45	-13.33%	7.2	2.38	8885	2909

Table C.1: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	101	106	-4.72%	10.52	4.9	8360	3758
graph_1	96	99	-3.03%	10.39	3.12	8394	2509
graph_2	102	105	-2.86%	9.76	5.41	7741	4259
graph_3	95	105	-9.52%	9.53	3.68	7629	2926
graph_4	98	102	-3.92%	10.01	4.96	8015	3917
graph_5	98	104	-5.77%	9.77	5.69	7967	4551
graph_6	99	103	-3.88%	10.05	4.53	7918	3606
graph_7	102	102	0.0%	9.89	8.2	7727	6392
graph_8	99	103	-3.88%	9.39	3.51	7499	2793
graph_9	95	99	-4.04%	11.29	5.98	8865	4654

Table C.2: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	158	162	-2.47%	13.44	10.5	8340	6509
graph_1	157	162	-3.09%	13.51	12.72	8374	7886
graph_2	155	159	-2.52%	12.84	7.6	7721	4572
graph_3	156	157	-0.64%	12.51	10.52	7609	6386
graph_4	150	153	-1.96%	12.86	10.0	7995	6214
graph_5	152	157	-3.18%	12.99	6.89	7947	4152
graph_6	150	153	-1.96%	12.7	8.0	7898	4970
graph_7	153	154	-0.65%	12.48	11.26	7707	6945
graph_8	152	155	-1.94%	12.08	6.86	7479	4231
graph_9	154	154	0.0%	14.45	13.07	8845	8007

Table C.3: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 50$.

Dual Barabási–Albert model - Random Parameters

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	32	56	-42.86%	24.02	10.44	30294	13030
graph_1	37	49	-24.49%	16.41	9.73	20465	12024
graph_2	33	55	-40.0%	29.57	5.1	36592	6349
graph_3	36	53	-32.08%	15.05	7.78	18806	9697
graph_4	48	51	-5.88%	144.21	35.78	176192	43126
graph_5	36	58	-37.93%	47.18	30.78	58984	38193
graph_6	37	45	-17.78%	24.76	4.91	31059	6113
graph_7	46	50	-8.0%	11.49	4.13	14455	5176
graph_8	33	51	-35.29%	33.55	4.0	42128	5004
graph_9	49	51	-3.92%	21.53	15.99	26948	19926
graph_10	40	52	-23.08%	17.53	8.81	21667	10666
graph_11	34	53	-35.85%	56.61	3.3	69759	4089
graph_12	43	44	-2.27%	13.68	10.14	17355	12851
graph_13	41	49	-16.33%	35.35	6.51	43263	8019
graph_14	46	46	0.0%	6.49	2.21	8163	2769

Table C.4: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	91	92	-1.09%	36.27	33.53	30274	27949
graph_1	99	113	-12.39%	24.46	15.22	20445	12527
graph_2	90	92	-2.17%	43.86	29.87	36572	25130
graph_3	97	97	0.0%	22.37	21.73	18786	18250
graph_4	75	83	-9.64%	213.13	72.71	176172	59854
graph_5	95	99	-4.04%	69.01	51.58	58964	44038
graph_6	89	97	-8.25%	37.31	24.94	31039	20567
graph_7	100	103	-2.91%	17.65	7.85	14435	6109
graph_8	94	99	-5.05%	50.99	34.65	42108	28592
graph_9	108	113	-4.42%	32.25	23.33	26928	19394
graph_10	98	102	-3.92%	25.41	18.88	21647	16020
graph_11	91	95	-4.21%	89.02	48.82	69739	38862
graph_12	97	104	-6.73%	20.66	9.37	17335	7804
graph_13	109	109	0.0%	53.98	47.5	43243	37897
graph_14	109	109	0.0%	9.69	8.0	8143	6705

Table C.5: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	151	153	-1.31%	49.7	49.06	30254	29857
graph_1	154	159	-3.14%	32.93	22.69	20425	14059
graph_2	154	159	-3.14%	60.71	49.81	36552	29830
graph_3	153	153	0.0%	31.09	28.34	18766	17061
graph_4	58	75	-22.67%	304.08	1.53	176152	932
graph_5	162	164	-1.22%	96.15	77.95	58944	47624
graph_6	141	148	-4.73%	49.86	26.3	31019	16361
graph_7	149	158	-5.7%	24.01	12.82	14415	7621
graph_8	160	161	-0.62%	69.12	54.55	42088	33245
graph_9	95	95	0.0%	46.11	40.38	26908	23564
graph_10	152	156	-2.56%	35.09	17.67	21627	10838
graph_11	158	172	-8.14%	115.55	62.97	69719	37737
graph_12	152	154	-1.3%	27.93	14.9	17315	9196
graph_13	55	65	-15.38%	75.29	0.0	43223	0
graph_14	149	159	-6.29%	13.14	10.27	8123	6329

Table C.6: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 50$.

Real-World Graphs

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	191	207	-7.73%	3.39	3.27	3895	3765
ca-netscience	65	65	0.0%	0.15	0.14	192	183
ia-email-univ	34	39	-12.82%	2.57	1.61	3256	2059
ia-infect-dublin	43	64	-32.81%	0.95	0.28	1236	365
inf-power	44	45	-2.22%	4.63	4.12	5676	5054
rt-retweet	21	21	0.0%	0.07	0.04	88	54
sc-shipsec1	204	220	-7.27%	900.0	121.11	281213	37811
soc-buzznet	27	45	-40.0%	900.0	252.2	383999	106859
socfb-CMU	43	82	-47.56%	900.0	108.31	1020899	124372
tech-RL-caida	26	53	-50.94%	900.0	555.81	214561	132658
tech-WHOIS	64	69	-7.25%	900.0	147.57	1008696	168031
tech-internet-as	31	46	-32.61%	88.58	30.48	69573	24002
tech-routers-rf	27	41	-34.15%	2.83	0.18	3446	219
web-arabic-2005	500	502	-0.4%	372.54	24.41	93433	6129
web-spam	44	66	-33.33%	53.91	2.04	64217	2464

Table C.7: The complete results of the evaluation on the real world graphs with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	380	382	-0.52%	5.19	5.01	3875	3745
ca-netscience	128	131	-2.29%	0.21	0.2	172	163
ia-email-univ	82	83	-1.2%	4.05	2.54	3236	2034
ia-infect-dublin	113	114	-0.88%	1.46	1.03	1216	868
inf-power	82	96	-14.58%	7.01	4.56	5656	3684
rt-retweet	33	43	-23.26%	0.08	0.01	68	2
sc-shipsec1	337	432	-21.99%	900.0	218.48	196485	47827
soc-buzznet	39	54	-27.78%	900.0	393.07	246995	106839
socfb-CMU	85	140	-39.29%	900.0	739.2	647201	529998
tech-RL-caida	62	75	-17.33%	900.0	793.6	150413	132638
tech-WHOIS	29	96	-69.79%	900.0	49.82	679691	37481
tech-internet-as	88	98	-10.2%	134.7	58.84	69553	30703
tech-routers-rf	68	84	-19.05%	4.26	0.21	3426	172
web-arabic-2005	1294	1420	-8.87%	497.4	63.22	93413	12100
web-spam	73	102	-28.43%	82.16	3.07	64197	2445

Table C.8: The complete results of the evaluation on the real world graphs with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	507	524	-3.24%	6.83	6.59	3855	3725
ca-netscience	185	185	0.0%	0.25	0.24	152	151
ia-email-univ	121	121	0.0%	5.3	4.05	3216	2454
ia-infect-dublin	122	124	-1.61%	1.93	0.69	1196	428
inf-power	146	148	-1.35%	9.25	8.19	5636	4993
rt-retweet	55	64	-14.06%	0.08	0.0	48	0
sc-shipsec1	549	578	-5.02%	900.0	302.68	148161	49884
soc-buzznet	39	61	-36.07%	900.0	602.58	185084	123727
socfb-CMU	159	209	-23.92%	900.0	367.69	501539	202106
tech-RL-caida	93	96	-3.12%	900.01	100.81	119957	13490
tech-WHOIS	60	104	-42.31%	900.0	66.49	502736	37439
tech-internet-as	48	101	-52.48%	185.85	3.52	69533	1337
tech-routers-rf	89	125	-28.8%	5.62	1.54	3406	941
web-arabic-2005	2488	2578	-3.49%	690.79	531.74	93393	72131
web-spam	132	138	-4.35%	110.07	62.91	64177	36634

Table C.9: The complete results of the evaluation on the real world graphs with $k = 50$.

C.1.2 Diversified top-k weighted clique search problem

Dual Barabási–Albert model - Same Parameters

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	264	277	-4.69%	7.04	1.91	8380	2271
graph_1	213	265	-19.62%	6.99	0.94	8414	1134
graph_2	262	262	0.0%	6.41	2.89	7761	3479
graph_3	246	260	-5.38%	6.43	2.0	7649	2345
graph_4	226	226	0.0%	6.78	2.32	8035	2741
graph_5	279	288	-3.12%	6.72	0.81	7987	782
graph_6	201	231	-12.99%	6.54	0.54	7938	653
graph_7	253	265	-4.53%	6.64	1.28	7747	1470
graph_8	251	251	0.0%	6.33	3.35	7519	3935
graph_9	231	317	-27.13%	7.43	2.96	8885	3535

Table C.10: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	567	567	0.0%	10.6	10.34	8360	8158
graph_1	525	567	-7.41%	10.61	3.14	8394	2478
graph_2	511	586	-12.8%	9.81	3.31	7741	2605
graph_3	497	587	-15.33%	9.63	3.71	7629	2926
graph_4	516	539	-4.27%	10.17	4.51	8015	3557
graph_5	584	601	-2.83%	10.18	6.72	7967	5212
graph_6	516	539	-4.27%	10.01	3.25	7918	2576
graph_7	541	602	-10.13%	9.92	4.56	7727	3505
graph_8	566	605	-6.45%	9.48	3.51	7499	2772
graph_9	548	583	-6.0%	11.25	3.77	8865	2977

Table C.11: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	861	861	0.0%	14.08	13.95	8340	8265
graph_1	804	865	-7.05%	14.33	10.34	8374	6040
graph_2	822	865	-4.97%	13.05	5.19	7721	3069
graph_3	801	812	-1.35%	12.9	6.92	7609	4080
graph_4	843	850	-0.82%	13.55	12.32	7995	7289
graph_5	836	839	-0.36%	13.87	10.67	7947	6080
graph_6	838	840	-0.24%	13.23	12.88	7898	7689
graph_7	892	906	-1.55%	12.91	10.04	7707	5997
graph_8	835	849	-1.65%	12.77	7.68	7479	4572
graph_9	805	845	-4.73%	15.1	8.24	8845	4834

Table C.12: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 50$.

Dual Barabási–Albert model - Random Parameters

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	286	337	-15.13%	26.4	11.71	30294	13448
graph_1	206	260	-20.77%	16.85	7.59	20465	9227
graph_2	282	307	-8.14%	30.48	15.66	36592	18752
graph_3	252	275	-8.36%	15.89	0.51	18806	613
graph_4	241	323	-25.39%	149.6	22.17	176192	26255
graph_5	313	367	-14.71%	48.77	34.69	58984	41918
graph_6	218	287	-24.04%	25.78	5.62	31059	6656
graph_7	277	295	-6.1%	11.94	3.98	14455	4779
graph_8	260	284	-8.45%	35.47	16.87	42128	20125
graph_9	257	257	0.0%	22.56	16.68	26948	19926
graph_10	259	274	-5.47%	18.26	8.92	21667	10317
graph_11	215	286	-24.83%	58.39	3.62	69759	4343
graph_12	204	240	-15.0%	14.42	2.96	17355	3540
graph_13	210	350	-40.0%	37.4	0.88	43263	1045
graph_14	236	243	-2.88%	6.72	1.59	8163	1937

Table C.13: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	492	525	-6.29%	38.21	21.66	30274	17071
graph_1	528	573	-7.85%	25.56	7.74	20445	6152
graph_2	502	540	-7.04%	48.17	38.64	36572	29656
graph_3	521	554	-5.96%	23.71	14.01	18786	11047
graph_4	326	425	-23.29%	228.42	0.56	176172	442
graph_5	528	559	-5.55%	75.07	55.46	58964	43621
graph_6	449	546	-17.77%	39.52	24.43	31039	19106
graph_7	472	592	-20.27%	18.28	5.72	14435	4489
graph_8	511	587	-12.95%	53.04	35.9	42108	28542
graph_9	629	655	-3.97%	34.21	26.5	26928	20872
graph_10	534	596	-10.4%	27.81	14.05	21647	10884
graph_11	501	541	-7.39%	88.7	44.09	69739	34594
graph_12	481	581	-17.21%	21.77	10.32	17335	8229
graph_13	290	323	-10.22%	57.13	14.77	43243	11455
graph_14	566	596	-5.03%	10.24	5.98	8143	4736

Table C.14: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	850	857	-0.82%	51.16	50.5	30254	29857
graph_1	874	895	-2.35%	34.48	28.22	20425	16685
graph_2	733	799	-8.26%	61.54	42.4	36552	25220
graph_3	798	849	-6.01%	31.7	20.76	18766	12269
graph_4	196	391	-49.87%	303.5	0.02	176152	11
graph_5	814	818	-0.49%	100.41	88.96	58944	52268
graph_6	811	860	-5.7%	52.54	39.09	31019	23064
graph_7	778	859	-9.43%	24.35	16.47	14415	9759
graph_8	601	636	-5.5%	71.39	7.02	42088	4187
graph_9	900	900	0.0%	45.65	41.84	26908	24672
graph_10	814	846	-3.78%	37.19	26.2	21627	15246
graph_11	229	367	-37.6%	118.66	0.02	69719	12
graph_12	791	877	-9.81%	29.73	13.79	17315	8075
graph_13	261	360	-27.5%	74.76	0.0	43223	1
graph_14	883	891	-0.9%	13.56	8.25	8123	4946

Table C.15: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 50$.

Real-World Graphs

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	1142	1249	-8.57%	3.83	3.69	3895	3765
ca-netscience	352	352	0.0%	0.15	0.15	192	183
ia-email-univ	158	197	-19.8%	2.78	1.7	3256	2032
ia-infect-dublin	370	371	-0.27%	1.08	0.55	1236	641
inf-power	196	196	0.0%	4.72	4.21	5676	5069
rt-retweet	142	142	0.0%	0.07	0.05	88	66
sc-shipsec1	1082	1151	-5.99%	900.0	158.17	280958	49924
soc-buzznet	90	190	-52.63%	900.0	3.72	384235	1537
socfb-CMU	238	328	-27.44%	900.0	680.83	1017588	769463
tech-RL-caida	143	332	-56.93%	900.0	559.33	212953	132650
tech-WHOIS	309	326	-5.21%	900.0	97.59	1009925	110874
tech-internet-as	190	231	-17.75%	90.64	42.16	69573	32380
tech-routers-rf	266	267	-0.37%	2.94	0.42	3446	493
web-arabic-2005	2706	2927	-7.55%	365.21	23.58	93433	6100
web-spam	218	373	-41.55%	55.82	2.12	64217	2464

Table C.16: The complete results of the evaluation on the real world graphs with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	2220	2293	-3.18%	5.94	5.73	3875	3745
ca-netscience	808	813	-0.62%	0.21	0.21	172	171
ia-email-univ	430	490	-12.24%	3.99	2.46	3236	2012
ia-infect-dublin	531	616	-13.8%	1.52	0.76	1216	621
inf-power	590	590	0.0%	7.31	6.53	5656	5062
rt-retweet	350	350	0.0%	0.08	0.07	68	57
sc-shipsec1	1740	2493	-30.2%	900.0	234.92	188322	49904
soc-buzznet	227	288	-21.18%	900.0	381.03	253840	106881
socfb-CMU	463	503	-7.95%	900.0	59.83	635415	42125
tech-RL-caida	438	495	-11.52%	900.0	796.11	149956	132630
tech-WHOIS	376	560	-32.86%	900.0	205.5	615418	147671
tech-internet-as	181	388	-53.35%	135.17	2.8	69553	1444
tech-routers-rf	353	461	-23.43%	4.38	0.19	3426	154
web-arabic-2005	7240	7792	-7.08%	564.19	36.16	93413	6213
web-spam	369	661	-44.18%	85.25	3.21	64197	2445

Table C.17: The complete results of the evaluation on the real world graphs with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	2930	2998	-2.27%	8.13	7.85	3855	3725
ca-netscience	1150	1150	0.0%	0.25	0.25	152	152
ia-email-univ	724	729	-0.69%	5.24	5.05	3216	3104
ia-infect-dublin	665	751	-11.45%	2.02	0.82	1196	485
inf-power	743	856	-13.2%	10.45	6.59	5636	3512
rt-retweet	447	462	-3.25%	0.08	0.07	48	43
sc-shipsec1	2960	3466	-14.6%	900.0	382.01	137663	58577
soc-buzznet	227	310	-26.77%	900.0	519.31	185889	106819
socfb-CMU	442	521	-15.16%	900.0	78.64	454513	42095
tech-RL-caida	615	641	-4.06%	900.0	843.48	117826	110540
tech-WHOIS	394	583	-32.42%	900.0	326.03	448021	167937
tech-internet-as	231	462	-50.0%	186.18	0.03	69533	9
tech-routers-rf	488	674	-27.6%	5.77	0.53	3406	322
web-arabic-2005	14480	14972	-3.29%	802.15	617.13	93393	72131
web-spam	640	881	-27.36%	115.58	4.21	64177	2430

Table C.18: The complete results of the evaluation on the real world graphs with $k = 50$.

C.2 DCCA-Mix

This subsection shows the results of DCCA-Mix. How we conducted the experiments for DCCA-Mix is the same as for DCCA-Same. Therefore, the explanation of each column is the same as in subsection C.1.

C.2.1 Diversified top-k clique search problem

Dual Barabási–Albert model - Same Parameters

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	24	36	-33.33%	6.83	0.17	8380	211
graph_1	18	31	-41.94%	6.78	0.82	8414	1019
graph_2	23	36	-36.11%	6.25	0.16	7761	197
graph_3	25	33	-24.24%	6.23	0.15	7649	182
graph_4	22	32	-31.25%	6.46	0.52	8035	641
graph_5	29	36	-19.44%	6.59	0.25	7987	119
graph_6	19	30	-36.67%	6.36	0.18	7938	226
graph_7	21	28	-25.0%	6.4	0.43	7747	495
graph_8	23	32	-28.12%	6.03	0.07	7519	87
graph_9	25	32	-21.88%	7.21	0.02	8885	24

Table C.19: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	88	89	-1.12%	9.81	6.37	8360	5425
graph_1	62	82	-24.39%	9.91	3.05	8394	2510
graph_2	94	94	0.0%	9.05	5.42	7741	4627
graph_3	73	84	-13.1%	9.04	5.87	7629	4926
graph_4	85	86	-1.16%	9.37	2.68	8015	2282
graph_5	100	100	0.0%	9.42	7.97	7967	6716
graph_6	90	90	0.0%	9.26	6.97	7918	5944
graph_7	91	91	0.0%	9.32	3.37	7727	2789
graph_8	91	91	0.0%	8.89	4.46	7499	3819
graph_9	100	100	0.0%	10.47	8.58	8865	7270

Table C.20: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	163	165	-1.21%	13.62	11.24	8340	6879
graph_1	148	151	-1.99%	13.67	8.06	8374	4933
graph_2	157	157	0.0%	12.6	10.91	7721	6680
graph_3	153	162	-5.56%	12.5	8.71	7609	5261
graph_4	159	159	0.0%	13.03	7.33	7995	4481
graph_5	158	158	0.0%	12.93	9.93	7947	6080
graph_6	151	154	-1.95%	12.85	8.09	7898	4970
graph_7	161	161	0.0%	12.35	12.29	7707	7673
graph_8	158	158	0.0%	12.16	8.74	7479	5372
graph_9	148	151	-1.99%	14.37	9.99	8845	6118

Table C.21: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 50$.

Dual Barabási–Albert model - Random Parameters

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	41	53	-22.64%	24.53	0.6	30294	747
graph_1	41	51	-19.61%	16.04	15.07	20465	19230
graph_2	48	55	-12.73%	29.29	6.41	36592	8047
graph_3	32	43	-25.58%	15.05	0.41	18806	458
graph_4	53	64	-17.19%	143.41	34.9	176192	43236
graph_5	44	52	-15.38%	47.25	14.17	58984	17669
graph_6	46	50	-8.0%	24.8	4.89	31059	6113
graph_7	32	43	-25.58%	11.49	0.48	14455	604
graph_8	44	52	-15.38%	33.85	0.72	42128	887
graph_9	30	38	-21.05%	21.64	11.48	26948	14355
graph_10	49	58	-15.52%	17.39	1.67	21667	1978
graph_11	55	57	-3.51%	55.66	26.55	69759	33230
graph_12	43	47	-8.51%	13.78	3.48	17355	4368
graph_13	43	49	-12.24%	35.52	0.84	43263	1042
graph_14	28	37	-24.32%	6.43	0.08	8163	96

Table C.22: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	109	114	-4.39%	36.27	16.21	30274	13486
graph_1	83	97	-14.43%	24.22	2.01	20445	1729
graph_2	115	116	-0.86%	43.66	35.02	36572	29271
graph_3	111	111	0.0%	22.44	19.45	18786	16332
graph_4	102	103	-0.97%	209.97	173.55	176172	145729
graph_5	115	120	-4.17%	70.37	65.06	58964	54419
graph_6	99	109	-9.17%	36.94	18.99	31039	16011
graph_7	69	98	-29.59%	16.98	7.48	14435	6414
graph_8	98	109	-10.09%	50.08	29.34	42108	24540
graph_9	50	60	-16.67%	32.7	0.02	26928	16
graph_10	106	114	-7.02%	25.3	10.88	21647	9329
graph_11	104	114	-8.77%	82.2	39.4	69739	33388
graph_12	102	104	-1.92%	20.47	7.37	17335	6165
graph_13	52	59	-11.86%	53.99	1.55	43243	1255
graph_14	53	62	-14.52%	9.81	0.1	8143	81

Table C.23: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	143	148	-3.38%	47.61	21.84	30254	13665
graph_1	158	165	-4.24%	31.44	18.47	20425	11984
graph_2	143	155	-7.74%	57.22	40.15	36552	25595
graph_3	153	155	-1.29%	29.35	19.14	18766	12254
graph_4	136	140	-2.86%	280.6	259.74	176152	163266
graph_5	157	158	-0.63%	92.27	74.71	58944	47524
graph_6	143	149	-4.03%	48.33	29.16	31019	18647
graph_7	154	154	0.0%	23.24	22.01	14415	13648
graph_8	151	152	-0.66%	65.48	51.39	42088	33016
graph_9	159	160	-0.62%	43.17	33.6	26908	20842
graph_10	159	159	0.0%	33.84	23.98	21627	15278
graph_11	152	162	-6.17%	111.51	59.27	69719	37576
graph_12	155	158	-1.9%	26.94	12.64	17315	8075
graph_13	150	151	-0.66%	70.19	58.17	43223	35601
graph_14	156	158	-1.27%	12.73	11.65	8123	7426

Table C.24: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 50$.

Real-World Graphs

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	191	208	-8.17%	3.38	3.26	3895	3765
ca-netscience	62	62	0.0%	0.15	0.14	192	179
ia-email-univ	34	40	-15.0%	2.66	1.65	3256	2059
ia-infect-dublin	42	54	-22.22%	1.01	0.3	1236	365
inf-power	28	39	-28.21%	4.59	4.07	5676	5038
rt-retweet	19	19	0.0%	0.07	0.03	88	37
sc-shipsec1	98	168	-41.67%	900.0	159.58	280791	49924
soc-buzznet	28	45	-37.78%	900.0	250.83	385146	107311
socfb-CMU	55	73	-24.66%	900.0	223.24	1050548	260384
tech-RL-caida	25	47	-46.81%	900.0	551.09	216080	132658
tech-WHOIS	51	68	-25.0%	900.0	0.83	1053040	759
tech-internet-as	36	41	-12.2%	88.52	71.37	69573	56227
tech-routers-rf	26	41	-36.59%	2.78	0.18	3446	219
web-arabic-2005	439	466	-5.79%	372.48	5.35	93433	1344
web-spam	44	52	-15.38%	53.73	2.03	64217	2464

Table C.25: The complete results of the evaluation on the real world graphs with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	380	393	-3.31%	5.3	5.11	3875	3745
ca-netscience	144	144	0.0%	0.2	0.2	172	171
ia-email-univ	89	91	-2.2%	4.06	2.48	3236	2012
ia-infect-dublin	97	107	-9.35%	1.51	1.06	1216	868
inf-power	115	115	0.0%	6.97	6.22	5656	5049
rt-retweet	51	53	-3.77%	0.08	0.06	68	46
sc-shipsec1	479	496	-3.43%	900.0	617.81	197041	135373
soc-buzznet	37	51	-27.45%	900.0	395.61	246570	107297
socfb-CMU	73	119	-38.66%	900.0	343.83	674661	260363
tech-RL-caida	65	88	-26.14%	900.01	793.76	150380	132638
tech-WHOIS	33	96	-65.62%	900.0	49.56	675493	37472
tech-internet-as	88	107	-17.76%	132.23	34.69	69553	18201
tech-routers-rf	67	84	-20.24%	4.25	0.24	3426	199
web-arabic-2005	1709	1759	-2.84%	500.35	384.98	93413	72151
web-spam	81	97	-16.49%	82.18	3.07	64197	2449

Table C.26: The complete results of the evaluation on the real world graphs with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	500	517	-3.29%	7.03	6.79	3855	3725
ca-netscience	207	207	0.0%	0.24	0.24	152	152
ia-email-univ	129	129	0.0%	5.2	3.96	3216	2454
ia-infect-dublin	123	128	-3.91%	1.97	0.79	1196	482
inf-power	146	160	-8.75%	9.31	8.25	5636	4998
rt-retweet	73	74	-1.35%	0.08	0.07	48	43
sc-shipsec1	554	582	-4.81%	900.0	776.62	145994	126013
soc-buzznet	39	56	-30.36%	900.0	482.15	185709	98743
socfb-CMU	173	250	-30.8%	900.0	482.06	490594	260343
tech-RL-caida	81	96	-15.62%	900.0	78.46	117373	10136
tech-WHOIS	55	88	-37.5%	900.0	1.53	504799	743
tech-internet-as	68	112	-39.29%	186.23	3.38	69533	1271
tech-routers-rf	91	118	-22.88%	5.81	1.32	3406	798
web-arabic-2005	2550	2550	0.0%	697.03	665.67	93393	89157
web-spam	143	146	-2.05%	111.68	40.38	64177	23557

Table C.27: The complete results of the evaluation on the real world graphs with $k = 50$.

C.2.2 Diversified top-k weighted clique search problem

Dual Barabási–Albert model - Same Parameters

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	153	223	-31.39%	6.79	0.17	8380	209
graph_1	137	208	-34.13%	6.81	0.75	8414	932
graph_2	132	199	-33.67%	6.28	0.12	7761	153
graph_3	154	179	-13.97%	6.26	0.23	7649	287
graph_4	120	152	-21.05%	6.5	0.52	8035	642
graph_5	166	243	-31.69%	6.54	0.22	7987	85
graph_6	119	185	-35.68%	6.37	0.19	7938	232
graph_7	155	215	-27.91%	6.29	0.42	7747	495
graph_8	174	208	-16.35%	6.05	0.11	7519	131
graph_9	202	245	-17.55%	7.27	0.74	8885	906

Table C.28: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	564	567	-0.53%	10.92	10.64	8360	8147
graph_1	526	536	-1.87%	10.74	7.34	8394	5707
graph_2	544	551	-1.27%	10.01	8.27	7741	6391
graph_3	498	551	-9.62%	9.64	5.17	7629	4100
graph_4	524	550	-4.73%	10.48	5.11	8015	3917
graph_5	625	669	-6.58%	10.42	6.86	7967	5192
graph_6	539	539	0.0%	10.28	8.91	7918	6865
graph_7	588	614	-4.23%	10.03	6.45	7727	4948
graph_8	566	585	-3.25%	9.68	3.87	7499	2990
graph_9	549	575	-4.52%	11.29	5.95	8865	4636

Table C.29: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	879	883	-0.45%	14.64	14.07	8340	8001
graph_1	850	850	0.0%	14.33	14.29	8374	8352
graph_2	807	849	-4.95%	12.97	9.72	7721	5778
graph_3	827	896	-7.7%	14.05	8.12	7609	4380
graph_4	823	823	0.0%	13.8	13.46	7995	7805
graph_5	878	878	0.0%	13.27	12.99	7947	7778
graph_6	839	848	-1.06%	14.11	13.78	7898	7717
graph_7	922	935	-1.39%	13.27	9.29	7707	5413
graph_8	883	883	0.0%	12.53	12.32	7479	7355
graph_9	830	872	-4.82%	15.66	8.48	8845	4834

Table C.30: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 50$.

Dual Barabási–Albert model - Random Parameters

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	179	284	-36.97%	25.31	10.11	30294	11904
graph_1	184	233	-21.03%	16.74	5.04	20465	6159
graph_2	169	258	-34.5%	30.58	14.61	36592	17240
graph_3	181	249	-27.31%	15.75	0.47	18806	559
graph_4	158	234	-32.48%	146.36	112.8	176192	135289
graph_5	209	322	-35.09%	48.59	29.31	58984	35197
graph_6	138	254	-45.67%	25.74	5.5	31059	6656
graph_7	163	272	-40.07%	12.03	3.27	14455	3941
graph_8	176	273	-35.53%	35.46	18.36	42128	21550
graph_9	209	258	-18.99%	22.66	0.38	26948	453
graph_10	227	258	-12.02%	18.43	4.58	21667	5257
graph_11	134	226	-40.71%	58.14	26.84	69759	32002
graph_12	216	261	-17.24%	14.48	2.44	17355	2954
graph_13	387	402	-3.73%	37.76	31.41	43263	36014
graph_14	137	214	-35.98%	6.74	0.09	8163	103

Table C.31: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	479	601	-20.3%	39.32	13.28	30274	10185
graph_1	533	571	-6.65%	25.86	8.64	20445	6815
graph_2	495	562	-11.92%	46.42	25.11	36572	19753
graph_3	507	604	-16.06%	24.02	14.78	18786	11467
graph_4	453	554	-18.23%	226.67	174.21	176172	135493
graph_5	506	609	-16.91%	74.69	49.5	58964	38913
graph_6	448	548	-18.25%	39.15	8.8	31039	6963
graph_7	481	586	-17.92%	18.25	6.27	14435	4951
graph_8	503	555	-9.37%	53.97	28.67	42108	22295
graph_9	326	336	-2.98%	34.68	0.06	26928	48
graph_10	525	592	-11.32%	27.78	9.37	21647	7263
graph_11	472	537	-12.1%	90.3	44.86	69739	34594
graph_12	528	571	-7.53%	21.97	11.03	17335	8690
graph_13	521	521	0.0%	57.03	49.65	43243	37655
graph_14	294	343	-14.29%	10.25	0.1	8143	81

Table C.32: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
graph_0	798	834	-4.32%	52.32	18.57	30254	10880
graph_1	863	935	-7.7%	35.65	22.94	20425	13169
graph_2	724	845	-14.32%	61.82	33.78	36552	19876
graph_3	837	862	-2.9%	32.74	19.41	18766	11418
graph_4	761	868	-12.33%	302.58	232.74	176152	136531
graph_5	849	894	-5.03%	100.3	82.85	58944	48547
graph_6	786	841	-6.54%	52.56	35.06	31019	21316
graph_7	817	833	-1.92%	24.9	10.98	14415	6394
graph_8	791	840	-5.83%	73.84	57.82	42088	33016
graph_9	662	673	-1.63%	47.53	41.61	26908	23672
graph_10	834	900	-7.33%	35.55	25.33	21627	15478
graph_11	802	861	-6.85%	117.75	60.64	69719	36878
graph_12	852	873	-2.41%	29.85	14.25	17315	8230
graph_13	920	920	0.0%	76.02	75.95	43223	43184
graph_14	795	795	0.0%	13.87	13.83	8123	8100

Table C.33: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 50$.

Real-World Graphs

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	1142	1239	-7.83%	3.95	3.81	3895	3765
ca-netscience	330	336	-1.79%	0.15	0.14	192	179
ia-email-univ	158	193	-18.13%	2.57	1.61	3256	2059
ia-infect-dublin	231	240	-3.75%	1.05	0.05	1236	55
inf-power	162	185	-12.43%	4.71	0.11	5676	137
rt-retweet	130	130	0.0%	0.07	0.05	88	66
sc-shipsec1	560	890	-37.08%	900.0	82.39	288251	26478
soc-buzznet	169	255	-33.73%	900.0	353.55	377722	148063
socfb-CMU	211	328	-35.67%	900.0	511.89	1013147	576615
tech-RL-caida	141	265	-46.79%	900.0	557.96	213625	132650
tech-WHOIS	316	327	-3.36%	900.0	97.72	1014030	110873
tech-internet-as	267	302	-11.59%	90.87	48.86	69573	37241
tech-routers-rf	225	225	0.0%	2.89	0.22	3446	264
web-arabic-2005	2098	2386	-12.07%	357.55	4.15	93433	1109
web-spam	218	342	-36.26%	55.37	2.11	64217	2464

Table C.34: The complete results of the evaluation on the real world graphs with $k = 10$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	2220	2293	-3.18%	6.1	5.87	3875	3745
ca-netscience	798	814	-1.97%	0.21	0.2	172	163
ia-email-univ	422	479	-11.9%	4.05	2.53	3236	2034
ia-infect-dublin	521	576	-9.55%	1.51	0.76	1216	621
inf-power	419	478	-12.34%	7.34	4.78	5656	3683
rt-retweet	202	270	-25.19%	0.08	0.01	68	4
sc-shipsec1	1771	2511	-29.47%	900.01	230.38	184694	47827
soc-buzznet	193	320	-39.69%	900.0	210.11	250692	58882
socfb-CMU	517	720	-28.19%	900.0	358.39	649487	260363
tech-RL-caida	364	528	-31.06%	900.0	761.66	148926	126196
tech-WHOIS	345	438	-21.23%	900.0	1.21	657896	759
tech-internet-as	390	592	-34.12%	140.35	35.21	69553	17508
tech-routers-rf	420	474	-11.39%	4.43	0.25	3426	198
web-arabic-2005	9099	9210	-1.21%	600.55	460.73	93413	72151
web-spam	369	639	-42.25%	86.33	3.23	64197	2445

Table C.35: The complete results of the evaluation on the real world graphs with $k = 30$.

Graph	Score End	Score Max	Difference Percentage	Run Time	Max Score Found Time	Total Step	Max Score Found Step
ca-GrQc	2893	2976	-2.79%	8.36	8.07	3855	3725
ca-netscience	1128	1137	-0.79%	0.27	0.25	152	143
ia-email-univ	677	694	-2.45%	5.78	5.45	3216	3027
ia-infect-dublin	665	749	-11.21%	2.0	0.8	1196	482
inf-power	743	843	-11.86%	9.87	8.75	5636	5000
rt-retweet	423	436	-2.98%	0.08	0.06	48	37
sc-shipsec1	3084	3256	-5.28%	900.0	500.11	141546	79259
soc-buzznet	235	326	-27.91%	900.0	610.66	182801	123727
socfb-CMU	784	1074	-27.0%	900.0	509.21	458015	260343
tech-RL-caida	541	590	-8.31%	900.01	858.91	115725	110532
tech-WHOIS	326	473	-31.08%	900.0	1.73	474851	743
tech-internet-as	697	936	-25.53%	196.25	50.19	69533	17491
tech-routers-rf	492	661	-25.57%	6.42	0.55	3406	322
web-arabic-2005	13860	14282	-2.95%	799.37	604.71	93393	71392
web-spam	715	767	-6.78%	121.69	4.54	64177	2431

Table C.36: The complete results of the evaluation on the real world graphs with $k = 50$.

C.3 TOPKLS and TOPKWCLQ

The results of TOPKLS and TOPKWCLQ are the mean results of five runs. In the score columns, we show the mean, standard deviation and the highest score of the five runs. The score is either the length of the coverage, for DTKC, or the summation of the

weights in the coverage, for DTKWC. The time found column, notes how long it took for the best scoring set was found in the run. Again, we state the mean and standard deviation. Lastly, we state the mean and standard deviation step this set was found. The results of the different cutoff times, 600 seconds and 60 seconds, will be seperated into different subsections.

C.3.1 Diversified top-k clique search problem (TOPKLS) with a cutoff time of 600 seconds

Dual Barabási–Albert model - Same Parameters

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	51.2	0.4	52	165.36	93.76	25159117.8	14282260.89
graph_1	48.4	0.49	49	206.43	176.14	29462900.2	24948414.37
graph_2	49.8	0.4	50	227.88	222.09	35840439.0	34938968.31
graph_3	52.0	0	52	243.88	135.65	36329709.6	19989814.39
graph_4	50.2	0.4	51	183.65	176.44	26152345.6	23909695.33
graph_5	54.2	0.4	55	186.14	167.07	27958128.8	25025605.08
graph_6	47.0	0	47	33.49	39.65	4768548.8	5558120.02
graph_7	46.4	0.49	47	32.88	27.39	4909558.6	4213456.18
graph_8	46.8	0.4	47	125.68	102.17	19183496.6	15688657.32
graph_9	52.4	0.49	53	71.59	78.29	10348784.8	11216604.95

Table C.37: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	110.0	0.63	111	290.29	205.98	33354386.0	23631787.59
graph_1	108.0	0	108	179.81	130.99	20557752.0	14581950.56
graph_2	108.4	0.49	109	128.18	120.41	15201800.8	14308033.88
graph_3	109.8	0.75	111	155.58	127.04	18360842.4	14984248.16
graph_4	109.0	0	109	107.2	70.07	12456872.2	8148009.68
graph_5	113.0	0	113	349.76	156.07	39800915.0	17776516.2
graph_6	106.4	0.49	107	137.59	136.06	15941719.2	15746009.69
graph_7	105.8	0.4	106	64.51	35.56	7662446.4	4225516.95
graph_8	105.0	0	105	87.01	77.35	10206079.6	9243104.1
graph_9	111.2	0.4	112	213.53	192.97	24812186.6	22469568.42

Table C.38: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	158.0	1.26	160	333.77	200.45	31222696.2	18489600.47
graph_1	159.8	0.98	161	231.72	165.14	21652276.4	15413220.01
graph_2	157.4	0.8	159	233.5	196.57	22903505.6	19252516.18
graph_3	154.2	0.4	155	256.28	219.74	24224329.6	20643037.98
graph_4	157.6	0.49	158	312.82	127.57	29278547.4	12025348.34
graph_5	160.8	1.47	163	263.6	216.23	24099234.6	20027091.97
graph_6	157.2	0.75	158	341.48	149.63	33267202.2	14608211.35
graph_7	155.8	0.4	156	167.95	77.39	16241257.0	7589953.67
graph_8	151.4	0.8	152	151.27	120.14	14960969.2	12061524.95
graph_9	159.4	1.02	161	244.11	193.23	22612961.8	17848551.42

Table C.39: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 50$.

Dual Barabási–Albert model - Random Parameters

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	68.6	0.8	70	217.57	167.29	27912751.0	21535654.06
graph_1	60.0	0	60	202.54	158.66	18945901.4	14951978.91
graph_2	70.0	0.63	71	255.76	104.47	33344455.8	13438415.31
graph_3	60.8	0.75	62	209.99	185.94	32349827.0	28733748.62
graph_4	85.2	1.17	87	160.02	109.65	18328167.8	12604613.25
graph_5	74.0	0	74	232.04	87.92	27436623.2	10357948.83
graph_6	59.6	0.8	61	221.99	168.15	19311842.6	14628465.51
graph_7	57.4	0.49	58	188.32	179.19	19115408.8	18198209.08
graph_8	68.8	0.4	69	341.58	212.32	42174808.6	25953460.46
graph_9	69.4	0.49	70	128.84	116.04	17897411.2	16138857.67
graph_10	65.4	0.49	66	250.25	119.76	26172796.6	12480249.46
graph_11	74.8	1.17	77	293.7	222.66	29675425.6	22515220.25
graph_12	57.0	0.63	58	140.7	79.49	17007277.8	9557722.38
graph_13	77.8	0.4	78	167.47	165.55	22344606.2	22111441.43
graph_14	54.6	0.49	55	207.36	184.53	29347513.8	26047919.84

Table C.40: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	130.2	0.75	131	316.51	156.35	32064693.2	15907812.28
graph_1	123.6	0.49	124	168.5	71.99	12868959.0	5405454.79
graph_2	133.2	0.4	134	106.85	62.8	10948838.0	6327827.05
graph_3	118.8	0.75	120	137.08	74.12	16276658.4	8753910.6
graph_4	149.6	1.02	151	214.26	164.96	19257165.6	14996233.9
graph_5	135.4	0.8	136	227.21	149.17	21160109.8	13895718.84
graph_6	125.8	0.75	127	348.89	200.97	25945983.6	14786223.57
graph_7	118.8	0.4	119	213.26	132.13	17552484.4	11051607.12
graph_8	129.2	0.4	130	242.28	109.66	23184137.6	10277507.04
graph_9	127.8	0.4	128	245.12	207.39	26089281.0	22062420.66
graph_10	126.6	1.02	128	251.79	130.47	21498930.4	11050025.86
graph_11	141.2	0.98	143	250.64	149.92	20225554.6	12125036.65
graph_12	117.6	0.8	119	158.78	137.93	15461854.0	13373144.7
graph_13	130.6	0.49	131	254.72	136.97	25856893.4	14077588.43
graph_14	113.4	0.49	114	190.64	127.06	20767250.8	13980533.93

Table C.41: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	184.8	0.98	186	267.08	142.61	22964535.4	12366324.48
graph_1	183.6	0.49	184	192.11	77.19	12832292.2	5184266.15
graph_2	187.2	0.75	188	281.62	202.95	24142138.2	17342541.22
graph_3	163.6	1.02	165	255.48	96.72	25491095.2	9699191.04
graph_4	202.0	0	202	204.8	175.98	16079726.4	13833596.85
graph_5	188.6	1.02	190	310.6	139.63	24989872.2	11321862.68
graph_6	185.2	0.75	186	199.0	171.52	12333560.4	10388134.02
graph_7	178.4	0.49	179	144.43	68.45	10408575.2	4902042.21
graph_8	183.0	0.63	184	199.92	178.31	16716021.6	14940293.77
graph_9	173.4	0.49	174	346.12	103.56	31634042.6	9498722.98
graph_10	184.8	0.4	185	289.53	210.11	21258930.8	15426216.78
graph_11	199.2	0.4	200	255.83	115.8	18355669.6	8442452.19
graph_12	175.4	0.49	176	171.35	128.43	14164858.0	10519846.98
graph_13	171.0	0.63	172	237.01	181.42	21747628.8	16738893.25
graph_14	162.6	1.02	164	160.52	205.74	14962149.6	19190171.72

Table C.42: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 50$.

Real-World Graphs

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	246.0	0	246	1.01	0.87	129570.6	105827.08
ca-netscience	68.0	0	68	0.02	0.01	2363.8	2101.33
ia-email-univ	75.0	0	75	188.42	211.65	31383322.0	35319254.31
ia-infect-dublin	110.0	0	110	0.22	0.16	20354.2	14019.4
inf-power	46.0	0	46	0.24	0.22	65687.6	64064.48
rt-retweet	24.0	0	24	0.01	0.0	716.0	386.6
sc-shipsec1	232.8	2.04	236	105.67	83.14	3745722.2	2959164.65
soc-buzznet	152.6	2.73	158	278.55	241.72	10937534.8	9466475.74
socfb-CMU	314.6	1.74	317	206.99	178.28	2934337.4	2520863.02
tech-RL-caida	103.4	1.02	105	344.41	164.46	58939271.2	27665261.85
tech-WHOIS	280.6	1.2	282	189.52	67.77	22230262.6	7777586.06
tech-internet-as	59.2	1.17	61	190.13	176.7	56976376.8	53085266.38
tech-routers-rf	95.4	0.49	96	247.66	60.79	54041863.4	13453852.17
web-arabic-2005	738.0	0	738	35.28	28.3	1625191.6	1306582.78
web-spam	126.0	0	126	175.13	89.22	20130436.2	10368411.23

Table C.43: The complete results of the evaluation on the real world graphs with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	422.4	0.49	423	151.41	195.92	14195683.6	18364234.21
ca-netscience	158.0	0	158	18.43	11.15	2308500.8	1383293.87
ia-email-univ	173.2	0.75	174	315.14	187.95	37204549.4	22330539.94
ia-infect-dublin	215.2	0.4	216	207.73	163.65	14110686.2	11344746.48
inf-power	120.8	0.75	122	225.83	94.23	39334972.6	16440476.55
rt-retweet	62.0	0	62	0.01	0.0	1195.0	477.32
sc-shipsec1	627.8	2.04	630	175.12	145.4	5024853.0	4208699.86
soc-buzznet	333.6	3.88	338	191.0	143.59	6263463.6	4649418.26
socfb-CMU	685.4	2.06	687	197.46	100.68	2461416.2	1289121.83
tech-RL-caida	242.0	2.28	246	241.14	101.47	27588969.6	11426598.96
tech-WHOIS	486.6	1.5	489	210.37	130.97	16808536.4	10364145.17
tech-internet-as	123.4	1.02	125	245.11	155.02	46597312.2	29461800.32
tech-routers-rf	180.4	0.49	181	392.7	131.95	56118451.4	18593684.91
web-arabic-2005	1769.0	0	1769	60.26	48.32	1624547.6	1306065.03
web-spam	267.6	1.5	270	211.22	128.4	18018180.4	10838497.6

Table C.44: The complete results of the evaluation on the real world graphs with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	546.6	0.49	547	237.28	151.16	17746747.2	11359490.11
ca-netscience	222.0	0	222	209.54	164.22	21045590.6	16323566.79
ia-email-univ	252.8	0.75	254	269.32	186.31	25500533.2	17699605.44
ia-infect-dublin	280.6	0.49	281	226.08	147.07	12441966.4	8104443.06
inf-power	180.8	0.75	182	287.5	120.03	39074560.6	16331639.41
rt-retweet	82.0	0	82	0.0	0.0	333.4	146.87
sc-shipsec1	997.8	2.71	1000	220.54	118.4	5509010.0	2957985.66
soc-buzznet	483.6	3.83	489	187.02	96.42	5837172.4	3016683.0
socfb-CMU	955.6	4.13	961	305.87	145.93	3753354.2	1796570.57
tech-RL-caida	362.4	2.06	366	356.33	158.03	30985744.2	13531116.58
tech-WHOIS	620.0	3.58	625	282.02	180.21	18569988.6	12021338.82
tech-internet-as	183.0	1.1	185	203.31	143.16	28507619.6	20152824.23
tech-routers-rf	246.8	1.33	249	297.91	122.24	34234775.4	14242307.54
web-arabic-2005	2789.0	0	2789	84.79	68.79	1623903.6	1305547.28
web-spam	382.0	2.1	385	203.92	127.82	14194164.4	8957428.32

Table C.45: The complete results of the evaluation on the real world graphs with $k = 50$.

C.3.2 Diversified top-k weighted clique search problem (TOPKW-CLQ) with a cutoff time of 600 seconds

Dual Barabási–Albert model - Same Parameters

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	346.6	3.2	350	368.14	108.19	45615830.6	13494287.21
graph_1	331.6	1.96	334	152.58	82.23	19264587.8	10381091.93
graph_2	344.6	2.15	348	322.13	180.85	42585000.2	23799599.53
graph_3	339.0	3.16	345	352.74	177.84	44695121.4	22664497.75
graph_4	333.6	1.96	337	310.83	243.55	39097626.8	30497689.39
graph_5	368.8	3.31	373	155.45	100.65	19124834.4	12005226.33
graph_6	327.4	1.74	329	215.01	164.44	27257290.2	21052928.69
graph_7	350.6	1.85	353	331.28	137.3	42670020.2	16908987.02
graph_8	343.8	1.6	346	357.99	143.34	46715114.6	18594978.5
graph_9	388.8	1.6	391	283.18	126.63	36759432.0	16472177.9

Table C.46: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	759.6	4.76	764	283.54	254.19	29465850.4	26470917.53
graph_1	763.2	2.99	768	337.69	84.96	34802840.2	8680970.73
graph_2	756.4	4.5	761	228.15	155.78	23649030.6	16049748.72
graph_3	737.4	1.85	739	412.82	196.18	43663597.4	20724037.2
graph_4	743.2	2.48	746	207.64	186.53	21296328.4	19109083.01
graph_5	772.8	3.12	777	198.16	112.7	20239128.6	11461290.54
graph_6	738.6	3.61	745	325.03	206.41	34355206.4	21838543.37
graph_7	776.6	2.15	780	386.81	162.17	41585447.2	17452270.45
graph_8	750.6	4.59	757	304.25	155.88	32655464.2	16737083.26
graph_9	794.4	6.22	803	331.21	119.12	35531396.6	12895900.2

Table C.47: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	1131.6	6.56	1143	167.07	135.82	13934632.0	10944692.97
graph_1	1147.2	3.66	1153	363.71	90.59	31523026.0	7982271.6
graph_2	1130.6	3.72	1134	296.94	173.54	26266947.8	15294900.26
graph_3	1106.0	2.37	1109	224.66	149.41	19848303.6	13237211.62
graph_4	1115.6	1.74	1118	167.85	132.15	14059418.2	10876078.47
graph_5	1144.2	2.48	1147	400.61	110.81	34127276.6	9289948.89
graph_6	1112.6	5.12	1120	306.99	133.44	26551267.6	11895389.22
graph_7	1156.0	3.52	1162	364.17	178.69	31135578.6	15394828.71
graph_8	1123.6	4.08	1131	209.58	152.01	18941429.6	13883159.85
graph_9	1166.6	6.28	1175	412.78	151.53	35742092.6	13201663.43

Table C.48: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 50$.

Dual Barabási–Albert model - Random Parameters

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	437.0	4.69	444	258.33	160.87	28606588.6	17973325.48
graph_1	390.8	3.54	394	378.75	68.78	30510095.4	5611380.62
graph_2	412.0	2.76	417	186.92	130.32	20528430.4	14220195.04
graph_3	374.6	2.65	378	234.4	118.17	31586610.6	16107019.26
graph_4	498.2	1.17	500	245.97	90.52	24642966.0	8921186.86
graph_5	483.6	4.59	490	345.21	168.36	35242337.0	17041588.95
graph_6	395.0	2.97	398	206.99	172.41	14670303.8	12365517.49
graph_7	364.8	2.14	369	409.84	175.54	35001266.2	14992101.32
graph_8	413.8	2.93	418	311.41	174.39	32306256.8	17868505.57
graph_9	415.4	2.42	418	299.72	182.44	35425276.2	21459743.01
graph_10	414.4	5.2	420	316.62	232.48	27881616.0	20542466.69
graph_11	434.2	3.82	440	356.97	102.07	30299364.2	8688129.2
graph_12	372.8	5.49	382	223.55	196.46	22331776.8	19381893.96
graph_13	481.2	2.32	484	261.02	95.07	31320353.8	11251397.5
graph_14	353.6	2.42	358	281.48	185.32	35070345.0	22891214.28

Table C.49: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	875.2	4.62	881	313.73	114.24	28865272.4	10384839.01
graph_1	862.4	6.15	874	372.94	80.86	26542060.0	5744423.86
graph_2	835.8	1.94	838	255.22	163.43	23879042.0	15299764.63
graph_3	782.2	3.82	786	253.37	164.41	28231534.6	18384280.1
graph_4	936.8	1.72	939	247.35	173.54	21098693.6	14805338.88
graph_5	919.6	3.98	923	473.46	88.78	40211125.6	7362610.37
graph_6	886.8	4.26	894	323.54	203.38	20524812.0	12900365.98
graph_7	810.8	2.48	814	312.88	175.69	22754423.4	12706900.53
graph_8	831.8	5.98	842	267.27	121.54	23967366.0	10916696.3
graph_9	821.6	2.33	824	296.35	145.86	29344323.2	14485415.22
graph_10	870.8	3.71	876	340.2	126.77	25753043.0	9572217.11
graph_11	898.8	6.24	910	254.92	213.51	18413164.4	15418907.99
graph_12	810.8	2.23	813	342.88	162.13	28126556.2	13402273.64
graph_13	870.4	4.22	875	291.66	195.47	29316506.8	19662374.07
graph_14	776.4	5.75	787	216.7	127.33	22489936.0	13205389.17

Table C.50: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	1263.0	6.72	1274	329.43	141.24	25656550.2	11066986.64
graph_1	1269.6	10.13	1289	314.01	203.61	19050321.0	12398337.86
graph_2	1222.0	4.98	1229	237.98	208.19	18879983.8	16660566.69
graph_3	1161.2	2.64	1164	442.26	79.24	41335879.6	7401937.36
graph_4	1323.4	4.41	1330	315.26	177.85	23190022.6	13134145.47
graph_5	1300.0	4.94	1306	433.23	110.98	31424986.8	8236580.03
graph_6	1314.2	3.71	1318	226.58	191.33	12295732.6	10484634.34
graph_7	1204.2	3.76	1209	268.71	49.83	16557439.4	2826521.28
graph_8	1213.6	3.83	1219	375.12	139.73	28536062.2	10826891.55
graph_9	1190.6	1.96	1193	336.29	165.97	28589315.2	14112715.49
graph_10	1268.0	3.95	1274	419.47	155.4	27058659.2	9911731.53
graph_11	1294.0	8.41	1308	294.87	165.87	18378440.8	10527114.11
graph_12	1203.4	2.87	1207	250.16	177.26	18310597.0	12921201.27
graph_13	1230.2	3.43	1235	268.62	138.37	22839257.2	11796120.07
graph_14	1154.8	6.4	1165	203.53	145.69	17560732.4	12632790.16

Table C.51: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 50$.

Real-World Graphs

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	1481.0	0	1481	0.84	0.71	111534.0	95679.26
ca-netscience	391.0	0	391	0.66	0.36	128154.0	69791.59
ia-email-univ	444.8	1.47	447	299.39	125.35	46769884.6	19486500.71
ia-infect-dublin	622.0	0	622	8.31	4.65	760307.0	439611.71
inf-power	286.0	0	286	1.23	1.42	344550.8	414999.14
rt-retweet	198.0	0	198	0.14	0.13	39388.4	39367.25
sc-shipsec1	1377.0	6.6	1389	264.92	128.89	9455755.8	4575869.56
soc-buzznet	904.4	15.29	914	348.69	147.5	12079882.4	5065507.21
socfb-CMU	1772.0	10.33	1790	230.32	174.28	3053837.0	2296305.69
tech-RL-caida	683.2	17.53	715	254.77	211.61	42835018.0	35313623.64
tech-WHOIS	1626.6	12.08	1640	206.82	21.87	24408257.2	2594237.57
tech-internet-as	391.8	11.72	413	328.23	185.64	102254382.0	58150547.69
tech-routers-rf	585.0	3.16	589	369.8	133.9	82197675.4	30358190.59
web-arabic-2005	4049.0	0	4049	51.43	48.54	2289660.6	2155222.25
web-spam	720.2	2.48	723	205.53	202.28	22332340.2	22130321.34

Table C.52: The complete results of the evaluation on the real world graphs with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	2511.2	1.47	2513	349.97	203.15	34304743.4	19930733.81
ca-netscience	947.6	0.8	948	240.02	203.06	32953758.2	27753192.13
ia-email-univ	1020.8	1.47	1022	146.99	148.11	17585244.4	17719673.28
ia-infect-dublin	1269.6	3.93	1276	206.28	172.99	14239918.4	11939151.94
inf-power	775.8	0.75	777	216.2	104.38	40638016.6	19643910.72
rt-retweet	434.0	0	434	3.1	3.16	556675.0	558365.63
sc-shipsec1	3797.2	16.07	3816	261.1	149.58	7697635.0	4387973.68
soc-buzznet	2016.6	27.36	2058	344.44	134.99	10781681.4	4255164.73
socfb-CMU	3931.0	14.18	3951	487.75	126.41	6168181.2	1559535.1
tech-RL-caida	1561.0	15.71	1581	111.67	68.12	13331238.8	8116599.46
tech-WHOIS	2807.8	12.81	2830	361.4	92.74	29326649.2	7621077.02
tech-internet-as	884.8	7.65	897	190.02	122.94	37246980.4	24215401.09
tech-routers-rf	1167.0	3.46	1172	182.56	95.34	28548356.2	14976109.46
web-arabic-2005	10483.0	0	10483	96.21	82.8	2470245.2	2132179.23
web-spam	1589.4	7.96	1597	313.61	186.47	26347815.8	15800238.11

Table C.53: The complete results of the evaluation on the real world graphs with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	3257.0	5.9	3264	428.65	119.72	31617116.6	8862658.48
ca-netscience	1304.2	0.75	1305	217.15	164.28	22144243.2	16725083.14
ia-email-univ	1490.6	3.98	1496	221.82	155.38	20462918.8	14418180.14
ia-infect-dublin	1662.0	1.26	1664	342.05	163.26	18680637.6	8939021.74
inf-power	1220.0	2.28	1224	97.08	25.17	13487207.0	3762415.27
rt-retweet	539.0	0	539	0.09	0.06	8377.0	6551.41
sc-shipsec1	6047.8	23.37	6079	304.24	170.21	7717723.8	4349463.96
soc-buzznet	2959.6	36.27	3012	380.19	152.12	10778817.4	4254034.39
socfb-CMU	5490.4	19.16	5515	282.29	229.39	3321747.8	2701505.98
tech-RL-caida	2346.4	15.37	2361	169.1	154.37	15029116.8	13725825.27
tech-WHOIS	3590.6	18.65	3621	190.77	173.9	12911780.4	11809759.83
tech-internet-as	1349.6	6.71	1362	292.23	176.77	42505847.6	25716458.6
tech-routers-rf	1630.6	4.96	1635	281.15	146.47	33548395.2	17421694.85
web-arabic-2005	16637.0	0	16637	127.09	110.03	2469265.2	2131333.34
web-spam	2271.2	7.88	2283	290.43	197.97	20535729.6	14106897.59

Table C.54: The complete results of the evaluation on the real world graphs with $k = 50$.

C.3.3 Diversified top-k clique search problem (TOPKLS) with a cutoff time of 60 seconds

Dual Barabási–Albert model - Same Parameters

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	50.2	0.4	51	15.11	15.63	2230962.2	2316914.21
graph_1	47.6	0.49	48	22.52	15.72	3220450.2	2322904.86
graph_2	49.0	0.63	50	12.31	11.7	1862785.6	1793180.4
graph_3	51.0	0	51	3.4	3.1	512093.8	469097.66
graph_4	49.2	0.4	50	11.39	16.42	1659551.4	2374018.86
graph_5	53.4	0.49	54	21.67	20.0	3160380.8	2918325.54
graph_6	46.8	0.4	47	11.03	9.15	1647563.6	1365134.46
graph_7	46.0	0.63	47	17.37	18.34	2651002.8	2805533.08
graph_8	46.2	0.4	47	12.74	15.0	1845338.6	2131540.88
graph_9	52.0	0.63	53	19.12	19.64	2865315.0	2945554.81

Table C.55: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	109.0	0	109	16.1	13.06	1902798.2	1542700.67
graph_1	107.2	0.4	108	21.0	19.01	2501331.6	2262281.68
graph_2	107.8	0.4	108	17.98	15.56	2182366.4	1895894.99
graph_3	109.2	0.98	111	11.44	6.93	1376443.0	832621.37
graph_4	108.4	0.49	109	17.19	17.54	2032839.0	2078177.69
graph_5	112.0	0	112	22.65	15.45	2646652.6	1807020.56
graph_6	105.8	0.75	107	23.02	20.02	2742733.6	2386708.08
graph_7	105.2	0.4	106	17.38	12.45	2103687.8	1504857.85
graph_8	104.4	0.49	105	6.75	5.32	824569.6	649770.72
graph_9	110.6	0.49	111	30.98	17.57	3624932.0	2004449.82

Table C.56: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	155.0	0.63	156	17.91	6.75	1784076.2	679268.15
graph_1	157.4	1.2	159	24.72	16.24	2467597.4	1621569.62
graph_2	154.8	1.83	157	28.92	22.15	2966398.4	2270906.29
graph_3	152.8	1.6	155	23.45	13.27	2400193.0	1357735.85
graph_4	155.2	0.75	156	30.98	5.48	3096391.0	541383.02
graph_5	159.0	1.9	162	26.47	14.0	2617776.6	1380868.53
graph_6	154.8	0.98	156	25.2	17.54	2526560.0	1762140.36
graph_7	153.2	0.4	154	29.52	17.53	2998878.4	1782085.25
graph_8	150.2	0.75	151	33.62	17.1	3477016.4	1768075.89
graph_9	158.2	1.47	161	35.16	6.4	3537580.2	646673.48

Table C.57: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 50$.

Dual Barabási–Albert model - Random Parameters

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	66.8	0.75	68	21.25	10.12	2759953.0	1330251.53
graph_1	59.0	0	59	15.25	12.05	1443283.0	1138980.43
graph_2	68.6	0.8	70	31.43	15.28	4143780.8	2016879.79
graph_3	60.0	0.89	61	23.69	5.89	3778244.6	924399.76
graph_4	83.6	0.8	85	12.74	9.61	1494262.8	1141648.9
graph_5	72.4	0.8	73	19.88	12.42	2332882.8	1439300.55
graph_6	58.4	1.74	61	18.05	10.92	1658974.8	1004713.26
graph_7	56.2	0.4	57	36.16	19.12	3629984.6	1925815.84
graph_8	67.4	1.02	69	18.16	12.74	2276661.8	1588855.41
graph_9	68.6	0.49	69	38.31	10.94	5530204.6	1573520.55
graph_10	63.4	0.49	64	17.53	8.15	1845449.6	856137.54
graph_11	73.0	0.89	74	27.8	20.37	2857206.0	2069577.66
graph_12	56.0	1.1	58	12.73	12.32	1492587.8	1461897.8
graph_13	77.4	0.49	78	36.32	4.44	4996526.0	600997.7
graph_14	53.6	0.8	55	12.61	9.17	1815145.8	1294592.4

Table C.58: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	127.8	0.75	129	26.1	18.67	2679800.0	1925751.88
graph_1	122.0	0	122	26.82	14.09	2071671.0	1096189.74
graph_2	132.0	0.63	133	21.0	11.59	2178688.6	1213273.36
graph_3	117.2	0.4	118	28.0	12.21	3391609.0	1555874.18
graph_4	147.8	1.33	150	29.43	9.63	2771689.6	936956.99
graph_5	134.2	0.75	135	21.15	16.63	1995368.4	1570574.62
graph_6	124.4	0.8	125	8.96	5.0	685845.4	384075.58
graph_7	117.8	0.4	118	36.7	9.5	3068274.0	775850.25
graph_8	127.6	0.49	128	11.24	10.16	1122360.8	1023971.44
graph_9	126.8	0.75	128	21.64	15.86	2351521.0	1723117.32
graph_10	124.4	0.8	125	19.45	16.28	1688334.0	1422589.28
graph_11	138.8	1.17	141	32.7	19.69	2773376.0	1673219.48
graph_12	116.6	1.62	119	9.82	15.58	978526.6	1550937.32
graph_13	129.2	0.75	130	33.5	20.91	3588249.0	2245670.22
graph_14	112.2	0.4	113	12.67	10.16	1385244.2	1078769.01

Table C.59: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	182.8	0.75	184	20.29	17.67	1807868.8	1577839.93
graph_1	181.2	0.4	182	23.18	17.91	1571529.6	1197193.48
graph_2	186.4	1.02	188	30.56	24.29	2590427.4	2034303.71
graph_3	161.6	0.8	163	30.23	18.96	3071774.8	1908375.09
graph_4	201.0	0.89	202	21.65	21.86	1692432.4	1713139.74
graph_5	186.2	0.4	187	15.24	11.51	1270205.2	967268.33
graph_6	184.4	0.8	185	10.24	5.73	674053.4	377469.97
graph_7	177.0	0.63	178	36.72	9.51	2676900.6	697740.98
graph_8	181.4	1.02	183	11.07	11.14	960395.8	969556.75
graph_9	171.2	0.98	172	15.04	13.44	1415592.6	1280855.21
graph_10	183.6	1.02	185	28.16	16.61	2142253.4	1268776.14
graph_11	198.0	0.63	199	26.35	21.47	1865407.2	1530076.88
graph_12	174.6	0.49	175	34.17	11.07	2920592.0	924280.93
graph_13	169.4	1.5	172	27.45	20.86	2550931.2	1951073.8
graph_14	160.4	2.06	163	14.21	9.82	1337476.2	922944.96

Table C.60: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 50$.

Real-World Graphs

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	246.0	0	246	0.94	0.75	129570.6	105827.08
ca-netscience	68.0	0	68	0.02	0.01	2363.8	2101.33
ia-email-univ	74.6	0.49	75	27.96	19.95	4736294.8	3377103.66
ia-infect-dublin	110.0	0	110	0.22	0.15	20354.2	14019.4
inf-power	46.0	0	46	0.23	0.22	65687.6	64064.48
rt-retweet	24.0	0	24	0.01	0.0	716.0	386.6
sc-shipsec1	230.6	2.06	234	22.69	12.61	831605.8	459314.64
soc-buzznet	147.6	2.8	151	20.79	23.19	849465.4	947780.45
socfb-CMU	310.2	3.92	314	27.4	17.06	408664.0	254107.83
tech-RL-caida	99.6	1.85	102	27.72	15.08	4878365.4	2655916.11
tech-WHOIS	276.8	1.17	278	27.51	16.45	3336670.8	2015445.31
tech-internet-as	56.6	1.5	58	28.43	13.15	8802101.2	4021959.21
tech-routers-rf	93.6	0.8	95	12.87	10.82	2932975.2	2470756.02
web-arabic-2005	737.8	0.4	738	22.94	18.48	1080079.2	872177.4
web-spam	124.4	0.49	125	28.82	15.82	3400250.0	1878020.18

Table C.61: The complete results of the evaluation on the real world graphs with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	422.0	0	422	16.81	12.66	1638312.6	1234839.84
ca-netscience	158.0	0	158	17.1	10.26	2308500.8	1383293.87
ia-email-univ	171.2	0.98	173	30.99	12.54	3765800.2	1532536.07
ia-infect-dublin	214.0	0.63	215	34.07	18.43	2443768.6	1323485.2
inf-power	119.4	0.49	120	20.93	8.78	3935742.0	1642382.44
rt-retweet	62.0	0	62	0.01	0.01	1195.0	477.32
sc-shipsec1	624.4	3.2	630	21.96	16.26	656103.2	486834.49
soc-buzznet	327.4	5.99	338	16.11	17.08	576281.2	613412.09
socfb-CMU	679.0	4.15	687	42.82	19.28	577896.8	258821.16
tech-RL-caida	233.8	1.94	236	27.66	9.64	3176637.6	1125775.04
tech-WHOIS	483.8	2.04	487	43.52	13.4	3659491.6	1109812.67
tech-internet-as	120.2	2.48	123	33.79	19.83	6635979.6	3854335.06
tech-routers-rf	178.4	0.49	179	33.2	18.11	5025531.2	2756907.05
web-arabic-2005	1768.6	0.49	1769	26.27	8.94	726506.0	248367.48
web-spam	263.8	0.98	265	26.22	13.92	2298260.2	1217366.2

Table C.62: The complete results of the evaluation on the real world graphs with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	544.6	1.36	546	16.33	15.26	1268528.0	1185200.68
ca-netscience	221.2	0.4	222	14.15	21.46	1463123.2	2212350.95
ia-email-univ	250.8	1.33	253	27.57	14.73	2681292.2	1442409.09
ia-infect-dublin	279.2	0.4	280	15.2	14.24	854225.4	793794.03
inf-power	179.4	0.49	180	27.84	11.55	3909686.0	1631509.83
rt-retweet	82.0	0	82	0.01	0.0	333.4	146.87
sc-shipsec1	993.6	3.2	999	33.85	11.09	861693.4	284503.9
soc-buzznet	477.4	6.59	489	34.14	19.86	1092520.0	629266.01
socfb-CMU	943.4	7.5	958	28.18	20.23	358160.8	257770.61
tech-RL-caida	353.8	1.94	356	35.61	12.52	3176129.6	1125595.0
tech-WHOIS	614.4	3.01	620	43.38	6.47	3008992.2	442662.68
tech-internet-as	179.2	1.94	182	27.47	14.62	3951492.8	2117264.15
tech-routers-rf	243.0	0.63	244	36.13	12.64	4271846.4	1479181.52
web-arabic-2005	2788.6	0.49	2789	36.26	12.4	726218.0	248269.02
web-spam	378.2	1.6	381	32.77	11.34	2355585.6	809906.22

Table C.63: The complete results of the evaluation on the real world graphs with $k = 50$.

C.3.4 Diversified top-k weighted clique search problem (TOPKW-CLQ) with a cutoff time of 60 seconds

Dual Barabási–Albert model - Same Parameters

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	340.2	1.94	342	30.56	21.19	3683393.6	2586101.66
graph_1	326.8	2.14	330	24.31	16.11	2969762.0	1974908.33
graph_2	339.2	1.94	342	20.82	13.2	2647113.4	1672653.43
graph_3	333.0	4.73	338	21.58	17.96	2615422.0	2184294.2
graph_4	331.6	2.8	337	32.26	17.2	3765645.8	1959382.5
graph_5	364.0	0.63	365	38.78	12.22	4647384.6	1535238.28
graph_6	324.2	2.04	328	17.93	13.99	2157489.8	1623778.68
graph_7	347.4	1.5	349	28.42	13.31	3560963.4	1673894.82
graph_8	334.4	1.2	336	20.0	19.99	2515936.6	2548720.71
graph_9	381.8	1.94	385	33.19	14.35	4128084.4	1835003.02

Table C.64: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	749.4	7.09	763	31.63	8.37	3230969.0	866307.39
graph_1	755.0	3.74	759	11.69	12.54	1118941.8	1135212.49
graph_2	750.0	6.48	759	41.58	14.07	4372632.8	1544689.09
graph_3	730.4	1.62	733	27.73	16.34	2796570.4	1711697.81
graph_4	740.6	2.87	745	28.64	19.14	2870563.0	1923809.55
graph_5	765.2	6.37	776	23.87	13.75	2311409.2	1316150.3
graph_6	731.2	1.17	733	25.31	21.21	2534296.0	2123584.72
graph_7	770.0	3.85	775	29.66	21.01	2904422.6	2014528.68
graph_8	745.2	6.05	757	26.46	20.14	2750389.0	2102986.93
graph_9	781.8	4.4	790	38.87	13.53	3950218.2	1393962.12

Table C.65: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	1123.0	6.36	1135	35.81	11.16	3153708.8	981171.39
graph_1	1137.6	5.78	1143	17.95	14.77	1577258.4	1294221.55
graph_2	1122.8	6.05	1133	30.43	11.33	2643978.8	969251.76
graph_3	1099.0	5.55	1109	38.25	19.82	3434973.0	1783450.07
graph_4	1113.2	1.94	1115	34.18	21.88	2959926.8	1922967.64
graph_5	1137.2	4.26	1144	27.41	13.19	2370368.6	1148120.9
graph_6	1102.6	4.63	1109	16.74	15.89	1433579.2	1380675.93
graph_7	1144.8	4.45	1152	35.38	15.63	3145089.0	1377469.44
graph_8	1118.0	6.87	1131	40.04	12.98	3673763.2	1180342.51
graph_9	1153.0	3.74	1157	36.76	15.1	3170191.8	1347827.59

Table C.66: The complete results of the evaluation on the generated graphs by the dual BA-model using the same input parameters with $k = 50$.

Dual Barabási–Albert model - Random Parameters

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	431.6	2.87	436	33.8	16.25	3706137.6	1787379.16
graph_1	384.2	2.48	389	34.11	12.39	2836562.2	1056025.88
graph_2	407.2	5.91	417	26.34	19.31	3038927.2	2244349.66
graph_3	368.8	4.07	374	17.58	11.4	2373778.2	1535664.29
graph_4	490.4	3.44	495	26.42	11.92	2811354.2	1264381.7
graph_5	479.0	7.92	489	29.23	17.86	3116357.8	1905738.36
graph_6	386.0	3.85	392	23.31	15.61	1752953.0	1174005.79
graph_7	357.8	3.43	361	38.26	17.13	3420531.4	1533505.33
graph_8	404.6	4.08	412	21.8	15.22	2267543.2	1570428.95
graph_9	411.0	3.35	417	47.3	11.75	5724360.6	1425517.89
graph_10	410.8	6.55	420	36.27	17.95	3237954.2	1607793.51
graph_11	425.6	4.41	433	33.71	21.93	2996466.8	1953788.1
graph_12	365.4	6.15	376	29.71	18.74	3169062.6	2005428.91
graph_13	475.0	4.0	481	35.77	13.15	4454373.8	1650956.74
graph_14	348.8	1.72	352	29.66	21.78	3753017.8	2760383.57

Table C.67: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	861.0	3.9	866	17.91	11.36	1680400.2	1069088.79
graph_1	856.8	5.27	866	36.18	15.32	2613148.8	1099143.48
graph_2	827.0	6.36	835	20.82	17.16	1903505.0	1568208.41
graph_3	774.8	4.96	784	35.23	9.52	3998043.4	1057810.52
graph_4	931.8	5.27	939	39.02	14.09	3249885.4	1138781.53
graph_5	909.0	4.94	916	27.11	11.89	2243383.0	1002590.43
graph_6	876.0	4.05	882	24.59	9.52	1525248.2	591130.97
graph_7	800.8	3.71	805	21.97	15.57	1530527.4	1080004.27
graph_8	821.8	3.31	826	30.72	15.17	2805019.0	1388632.52
graph_9	814.8	4.71	822	19.19	11.87	1931614.8	1211252.76
graph_10	859.4	3.93	865	37.64	15.5	2945906.4	1218525.61
graph_11	888.0	3.35	894	24.19	14.95	1810397.6	1108344.01
graph_12	803.8	5.15	808	32.26	15.49	2737688.2	1347450.58
graph_13	864.4	1.02	866	37.86	17.14	3718504.6	1696077.43
graph_14	764.4	1.02	766	29.4	15.21	3089665.4	1604001.71

Table C.68: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
graph_0	1248.8	5.04	1255	22.71	15.34	1767470.2	1207186.89
graph_1	1253.8	10.09	1270	22.53	10.01	1326228.0	596734.24
graph_2	1215.8	6.73	1226	21.9	17.17	1764883.6	1402876.68
graph_3	1151.6	3.5	1158	46.46	8.76	4515394.8	857922.09
graph_4	1313.8	8.77	1330	25.85	19.64	1893581.0	1433647.74
graph_5	1290.2	7.49	1302	23.27	10.84	1714274.0	794268.74
graph_6	1306.4	6.62	1316	19.94	8.23	1081699.8	455235.64
graph_7	1192.4	4.27	1198	20.58	17.38	1301662.6	1061197.06
graph_8	1204.0	2.61	1206	44.11	7.65	3500015.4	600118.77
graph_9	1183.4	6.28	1191	35.14	15.37	2802986.8	1271819.25
graph_10	1258.4	5.12	1263	31.42	13.07	2140566.6	894826.05
graph_11	1281.2	5.34	1290	28.78	17.51	1823506.0	1099356.25
graph_12	1195.2	4.75	1203	39.99	17.87	2965493.4	1324173.47
graph_13	1223.6	0.8	1225	31.14	16.02	2764665.2	1431713.57
graph_14	1140.4	3.88	1145	43.88	8.2	3919830.4	730050.58

Table C.69: The complete results of the evaluation on the generated graphs by the dual BA-model using a range of input parameters with $k = 50$.

Real-World Graphs

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	1481.0	0	1481	0.79	0.68	111534.0	95679.26
ca-netscience	391.0	0	391	0.63	0.34	128154.0	69791.59
ia-email-univ	440.6	1.74	443	24.4	18.08	4008789.6	2979608.81
ia-infect-dublin	622.0	0	622	7.95	4.58	760307.0	439611.71
inf-power	286.0	0	286	1.16	1.41	344550.8	414999.14
rt-retweet	198.0	0	198	0.13	0.12	39388.4	39367.25
sc-shipsec1	1366.2	4.26	1371	35.78	15.65	1333950.8	584272.47
soc-buzznet	847.8	15.48	863	33.34	11.77	1204923.8	428684.3
socfb-CMU	1751.0	24.73	1790	39.3	11.47	564352.4	163731.4
tech-RL-caida	661.6	13.63	678	45.82	17.33	7787935.6	2947499.04
tech-WHOIS	1587.2	14.03	1609	31.69	15.1	3865606.8	1827879.96
tech-internet-as	367.4	8.14	380	30.16	11.95	9662682.8	3796481.9
tech-routers-rf	578.4	4.32	582	40.43	13.87	9292910.4	3175704.03
web-arabic-2005	4047.0	2.45	4049	10.1	12.88	474041.8	603241.11
web-spam	715.6	5.71	723	34.49	15.59	3998794.0	1804361.56

Table C.70: The complete results of the evaluation on the real world graphs with $k = 10$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	2505.4	4.96	2512	22.67	15.66	2251287.6	1555626.84
ca-netscience	945.6	2.06	948	11.04	12.89	1534781.8	1786190.36
ia-email-univ	1017.6	3.93	1022	30.92	16.24	3756027.8	1959364.49
ia-infect-dublin	1262.6	1.85	1266	27.14	18.67	1961036.0	1349856.81
inf-power	772.8	0.75	774	26.36	12.75	5231112.4	2542945.15
rt-retweet	434.0	0	434	2.86	2.86	556675.0	558365.63
sc-shipsec1	3768.8	15.66	3793	34.15	13.8	1038995.2	420512.55
soc-buzznet	1952.4	21.96	1984	25.31	5.33	813088.6	167719.28
socfb-CMU	3888.0	14.17	3909	36.17	17.87	476620.8	235157.16
tech-RL-caida	1531.4	16.7	1551	15.21	12.29	1774246.0	1412637.81
tech-WHOIS	2766.6	9.77	2775	29.75	15.77	2531547.2	1324022.45
tech-internet-as	866.6	11.48	886	27.1	24.1	5574826.0	4943954.9
tech-routers-rf	1156.8	8.63	1170	27.51	17.9	4391034.8	2863130.6
web-arabic-2005	10476.2	6.05	10483	20.41	18.58	574653.8	523491.72
web-spam	1571.4	13.75	1594	13.51	3.84	1206762.2	348472.43

Table C.71: The complete results of the evaluation on the real world graphs with $k = 30$.

	Scores			Time Found		Step Found	
Graph	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Mean**	**SD**
ca-GrQc	3245.6	2.15	3249	25.96	10.94	2082723.6	866788.22
ca-netscience	1302.0	0.89	1303	30.91	15.77	3338243.2	1702906.76
ia-email-univ	1480.6	4.32	1486	19.79	16.14	1944265.8	1587032.76
ia-infect-dublin	1655.8	2.93	1659	36.77	19.43	2084776.0	1101377.54
inf-power	1212.2	2.04	1216	19.12	7.21	2825183.6	1067069.32
rt-retweet	539.0	0	539	0.08	0.06	8377.0	6551.41
sc-shipsec1	5997.6	19.06	6028	31.19	18.65	806849.8	485853.83
soc-buzznet	2892.0	47.68	2960	23.24	11.63	662345.6	338013.57
socfb-CMU	5444.6	36.1	5497	23.08	19.59	285664.4	241937.93
tech-RL-caida	2315.2	23.58	2351	31.81	19.01	2932011.8	1785989.61
tech-WHOIS	3557.4	23.84	3589	25.5	9.98	1783790.2	698120.93
tech-internet-as	1320.8	9.04	1331	28.92	23.02	4385152.8	3498676.33
tech-routers-rf	1614.6	10.8	1633	22.18	20.25	2747483.4	2510840.01
web-arabic-2005	16628.4	7.09	16637	17.9	19.9	362747.6	403562.7
web-spam	2254.4	5.89	2262	31.4	18.96	2336738.8	1412738.35

Table C.72: The complete results of the evaluation on the real world graphs with $k = 50$.