

Searching for a k -Clique in Unknown Graphs

Roni Stern

Information Systems Engineering
Ben Gurion University
Beer-Sheva, Israel
roni.stern@gmail.com

Meir Kalech

Information Systems Engineering
Ben Gurion University
Beer-Sheva, Israel
kalech@bgu.ac.il

Ariel Felner

Information Systems Engineering
Ben Gurion University
Beer-Sheva, Israel
felner@bgu.ac.il

Abstract

Agents that solve problems in *unknown graphs* are usually required to iteratively explore parts of the graph. In this paper we research the problem of finding a k -clique in an *unknown graph* while minimizing the number of required exploration actions. Two novel heuristics (*KnownDegree* and *Clique**) are proposed to reduce the required exploration cost by carefully choosing which part of the environment to explore. We further investigate the problem by adding probabilistic knowledge of the graph and propose an Markov Decision Process(MDP) and a Monte Carlo based heuristic (*RClique**) that uses knowledge of edge probabilities to reduce the required exploration cost. We demonstrate the efficiency of the proposed approaches on simulated random and scale free graphs as well as on real online web crawls.

Introduction

Many real-life problems can be modeled as problems on graphs, where one needs to find subgraphs that have a specific structure or attributes. Examples of such subgraph structures are shortest paths, any path, shortest traveling salesperson tours, minimum spanning trees and cliques. Most classic algorithms that solve those graph problems assume that the entire structure of the graph is given as input in data structures (e.g. adjacency list or adjacency matrix). We refer to such problems as problems on **known graphs**.

By contrast, there are real-life problems which can be modeled as problems on graphs where the graph is not given as input and is not necessarily known in advance. For example, a robot navigating in an unknown terrain where vertices and edges are physical locations and roads respectively. Another example is an agent searching the World Wide Web, where the vertices are the web sites and the edges represent hypertext links. We refer to such problems as problems on **unknown graphs**.

Solving problems in such **unknown graphs** requires exploring some part of the graph. We define an exploration action, that when applied to a vertex, discovers all its neighboring vertices and edges. In the web graph domain, for example, the exploration corresponds to sending an HTTP request, retrieving an HTML page and parsing all the hypertext links in it. The hypertext links are the outgoing edges,

and the connected web sites are the neighboring vertices. For a physical domain, the exploration is applied by using sensors at a location to discover the near area, e.g., the outgoing edges and the neighboring vertices on the map.

Such exploration actions are associated with a cost. The exploration cost often requires a different resource than the traditional computational effort. For the web graph scenario, the exploration cost includes network I/O of sending and receiving IP packets. For physical world environment, exploration cost is associated with the cost of activating sensors at a vertex. In such cases, an important task will be to solve the problem while minimizing the exploration cost. Computational CPU cost can be therefore neglected as long as it is running in polynomial time.

Cliques in Unknown Graphs

In this paper we focus on the problem of searching for a k -clique in unknown graphs, starting from a single known vertex. A k -clique is a set of k vertices that are pairwise connected. The task is to find a k -clique with minimum exploration cost. Beyond the value of the investigation of such a basic problem in unknown graphs, k -clique can be practical in real-world unknown graph domains. For example, finding a set of physical locations forming a clique suggests the existence of a metropolitan. Another example is an agent searching for a set of web sites forming a clique. A clique of web sites suggests resemblance in content. Consider, for example, a searching process of an online academic database such as Google Scholar and CiteSeer. Each paper is referenced by a hypertext link, and citations of a paper are also available as links. Assume a clique of papers is found, i.e. a group of papers where each paper cites or is cited by all the other papers in the group. It is very likely that all papers in the group discuss the same subject. Thus finding a clique in such a web site allows a focused content search.¹ A possible application is a web crawler searching an online database such as CiteSeer or Google Scholar (many of which can not be completely downloaded).

Most classic algorithms for finding cliques (Bron and Kerbosch 1973; Tomita and Kameda 2007; Pullan and Hoos 2006) are applied to *known graphs*. For example, some of

¹This structural approach can of course be complemented by text data mining approach to prune irrelevant papers.

these algorithms order the vertices by their degrees and then start by accessing the vertex with the highest degree. Such algorithms are not naturally applicable to unknown graph, since in unknown graphs the degree of the vertices is unknown. There has been some preliminary work by Altshuler et. al. (2005) on searching for a k -clique in unknown *physical* graphs. However, the focus of that work was on different communication paradigms for multiple agents that search for a k -clique in physical unknown graphs.

We introduce a general framework for finding a k -clique in an unknown graph based on Best-First Search. This algorithm uses a heuristic to choose the next vertex to explore. We present several such heuristics: The *KnownDegree* heuristic which considers the highest degree of vertices on the explored part of the graph. The *Clique** heuristic which chooses vertices that are connected to a larger *potential* k -clique. We propose two options to exploit probabilistic knowledge of the graph, if such exists. An MDP formalization is presented, that allows calculating an optimal exploration policy and a more practical Monte-Carlo based heuristic, named *RClique**.

We evaluated the exploration cost of the presented algorithms empirically on random graphs and scale free graphs (Eppstein and Wang 2002; Bonato and Laurier 2004). Additionally, we applied the different algorithms as part of an online web crawler, searching for cliques of citations in *Google Scholar*. All the above algorithms as well as a baseline random exploration algorithm were compared. Results show that all algorithms are superior to the random algorithm requiring less than 50% of the exploration cost. *Clique** is shown to be usually better than *KnownDegree*, and *RClique** is able to use the knowledge of the edge probabilities to significantly outperform *Clique**.

Problem Definition

The following definitions are required for the k -clique problem in unknown graphs.

Definition 1 [Known Graph] Given a graph $G = (V, E)$ and an agent a , the known graph $G_{known} \subseteq G$ represents the part of the graph that is known to a , where $V_{known} \subseteq V$ represents the known vertices and $E_{known} \subseteq (V_{known} \times V_{known}) \cap E$ represents the known edges.

We define an exploration action for the agent which can be operated on any vertex $v \in V_{known}$. This exploration model is inspired by the *fixed graph* model (Kalyanasundaram and Pruhs 1994).

Definition 2 [Explore] The action *Explore*(v) adds the edges connected to v to E_{known} , and adds the neighboring vertices of v to V_{known} .

We call two vertices *neighbors* if there is an edge connecting them in G . We define the function *neighbors*(v) to return the set of neighboring vertices of vertex v . For ease of notation, we borrow the terms *expanded* and *generated* from the classic search terminology in the following way. Applying an exploration action to a vertex will be referred to as expanding that vertex and generating its neighbors. We denote $V_{exp} \subseteq V_{known}$ as the set of all the vertices that have been

expanded and $V_{gen} \subseteq V_{known}$ as the set of all the vertices that have been generated but not expanded. A typical vertex goes through the following stages. First it is unknown. Then when it is generated, it is added to V_{known} . Finally, when it is expanded, its incident edges and its neighboring vertices are added to G_{known} .

Each exploration action has a corresponding exploration cost which depends on the vertex:

Definition 3 [Exploration Cost] The function *exCost* : $V \rightarrow \mathbb{R}$ obtains a vertex and returns its exploration cost.

The cost function is additive, meaning that the total cost of multiple exploration actions is the summation over all the exploration costs of the explored vertices. In this paper we assume a constant exploration cost C : $\forall v \in V \text{ exCost}(v) = C$ for some C . This is motivated by a number of real world scenarios such as the following examples. (1) a central controller that can be queried to provide the exploration data. (2) a robot that senses its nearby environment where the sensing cost is constant. (3) querying a web page from a single host. For example, we have measured the runtime required to query Google Scholar and parse the resulting web page for its hypertext links.² More than 90% of the queries required approximately the same amount of time (+200 milliseconds). Without loss of generality, we will assume in the rest of the paper that $C = 1$ for simplicity. Now we can define the k -clique problem in an unknown graph:

Definition 4 [k -clique problem in an unknown graph] Assume a graph $G = (V, E)$, a constant k and an agent a with the known graph $G_{known} \subseteq G$, where $V_{gen} = \{s\}$, $V_{exp} = \emptyset$ and $E_{known} = \emptyset$. Armed with *Explore* action, the goal of a is to find a k -clique in G with a minimal exploration cost.

Best-First Search Approach

Searching for a k -clique in unknown graphs is inherently an iterative process, in which the graph is explored vertex by vertex. Therefore, we propose a best-first search approach. In every step the agent chooses the next target vertex to explore from V_{gen} . After exploring each vertex, we search for a k -clique in G_{known} . A simple brute force implementation of a k -clique search in G_{known} has a worst case computational complexity of $O(|V_{known}|^k)$. For small values of k this is tractable. Note that this search for a k -clique does not require any exploration cost, since G_{known} contains the part of the graph that has already been discovered. This loop of selecting a vertex to explore, exploring it and testing the known graph for a k -clique continues until a k -clique is found or when all the vertices in the graph are explored. This is equivalent to a best-first search where V_{gen} is the *open-list* and V_{exp} is the *closed-list*.

The main challenge is how to choose the next vertex to explore. In the worst case, all the vertices must be explored (as explained above). However, we show that in practice, an intelligent heuristic that determines the "best" vertex to

²Data was gathered from a single end-user running 2,400 different queries.

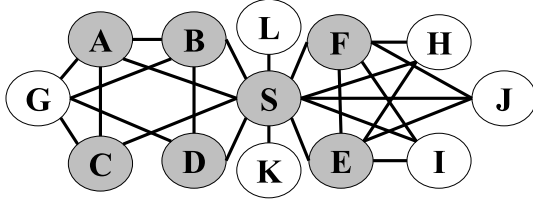


Figure 1: Example of the different heuristics.

explore next can greatly reduce the number of explored vertices. Such a heuristic should exploit the current knowledge of the graph - G_{known} - to determine which of the vertices in V_{gen} has highest chances to lead to the closest k -clique. Next we propose three such heuristics.

Non-Probabilistic Heuristics

Next we introduce two heuristics *KnownDegree* and *Clique** which assume that only the initial vertex of the graph is given. In the following section we propose a heuristic which utilizes probabilistic knowledge about the graph if such is available.

Known Degree

A very common and effective heuristic used for the k -clique problem in known graphs is to search vertices with high degree (Tomita and Kameda 2007). Vertices with a high degree are more likely to be a part of a k -clique, than vertices with a lower degree (Bollobas and Erdos 1976). In addition, since vertices with a higher degree have more neighboring vertices, exploring them adds more edges and vertices to G_{known} . Since the real degree of a vertex in V_{gen} is not known as it has not been explored yet, we consider its *known degree* as the number of *expanded* vertices that are adjacent to it. The *KnownDegree* heuristic chooses to explore the vertex with the highest known degree in V_{gen} .

For example, consider the known graph in Figure 1. The grayed vertices are the explored vertices, and the white vertices are the generated vertices (from which to choose the next vertex to explore). Using *KnownDegree*, the next vertex that will be explored is G , since its known degree (=4) is the highest among the generated vertices.

Clique*

The second heuristic relates to the size of the largest clique that can be extended to a k -clique. This is called a *potential k -clique* which is defined next along with supporting terms. We then develop a heuristic that selects to explore the vertex connected to the maximal potential k -clique.

[RONI:MODIFIED DEFINITION]

Definition 5 [Potential k -Clique] A set of vertices $V_{PC} \subseteq V_{exp}$ where $|V_{PC}| < k - 1$ is a *potential k -clique* if there exists a graph $G' = (V, E)$ that can be extended from G_{known} (i.e., $G_{known} \subseteq G'$) and V_{PC} is part of a k -clique in G' .

We define the following function that returns a set of vertices in V_{gen} such that each member of this set is a neighbor of every vertex in V_{PC} .

Definition 6 [Generated Common Neighbors]

Let V_m be a set of expanded vertices. Then $gcn(V_m) = \bigcap_{v_i \in V_m} neighbors(v_i) \cap V_{gen}$.

These definitions lead to the Corollary 1. We omit a formal proof of this Corollary due to space limitations. The proof is based on the fact that if a group of vertices is a k -clique then any subset of that group must also be a clique.

Corollary 1 A set of vertices $V_m \subseteq V_{exp}$, where $|V_m| = m < k - 1$ is a *potential k -clique* if and only if (1) V_m is an m -clique and (2) $|gcn(V_m)| \geq k - m$.

For every vertex $v \in V_{gen}$ we denote $\mathcal{PC}_k^v$ as the set of all potential k -cliques V_{PC} such that $v \in gcn(V_{PC})$. Let $\overline{V_{PC}}$ be the largest potential clique among $\mathcal{PC}_k^v$ and assume $|\overline{V_{PC}}| = x$. If $\overline{V_{PC}}$ is indeed a part of a k -clique in G then in the best case all we need is to expand $k - 1 - x$ vertices to reveal the k -clique. This enables defining the following cost function $h_{c*}(v)$ for every $v \in V_{exp}$ which estimates the remaining cost to complete a k -clique.

Definition 7 [The Clique* cost function]

$$h_{c*}(v) = \min_{V_{PC} \in \mathcal{PC}_k^v} k - 1 - |V_{PC}|$$

The *Clique** heuristic chooses to explore the vertex with the lowest h_{c*} from V_{gen} .³ Ties are broken using the *KnownDegree* heuristic. Theorem 1 states that if vertex v is chosen by *Clique** then $h_{c*}(v)$ is a *tight lower bound* on the cost of exploring a k -clique.

Theorem 1 *Clique** is an *admissible heuristic*. There is no better *admissible heuristic*, i.e. if vertex v is chosen by *Clique** then there is no *admissible function* h' such that $h'(v) > h_{c*}(v)$.

Proof: Let V_m be a set of m vertices that are a potential k -clique. Extending V_m to a k -clique requires connecting all its vertices to $(k - m)$ vertices that form a $(k - m)$ -clique and have not been explored yet. Exploring a $(k - m)$ -clique requires a minimum of $k - m - 1$ exploration actions. Thus in the best case extending V_m to a k -clique requires an exploration cost of $k - m - 1$. Since a vertex $v \in V_{gen}$ may be connected to several potential k -cliques, a lower bound on the exploration cost of finding a k -clique if v is explored, is the minimum over the lower bound of the exploration cost of extending each of the potential k -clique in $\mathcal{PC}_k^v$. By definition, every potential k -clique can be extended to a k -clique in a graph that is consistent with G_{known} (see Definition 5). Therefore after exploring a vertex v , a k -clique may be explored in G after $h_{c*}(v)$ exploration steps. Thus no better *admissible heuristic* can be created. $\square$

In order to use *Clique**, $\mathcal{PC}_k^v$ must be computed for every vertex in V_{gen} before choosing the next vertex to explore. This can be done by maintaining a global list of potential k -cliques, denoted by $\mathcal{PC}_k$. Every set of expanded vertices that form a potential k -clique is stored in $\mathcal{PC}_k$. When a vertex is expanded, $\mathcal{PC}_k$ must be updated as potential k -cliques may be created or deleted. Expanding a node with degree D this will require updating $\sum_{i=1}^{k-1} \binom{D}{i} \approx O(D^{k-1})$ potential

³If $\mathcal{PC}_k^v$ is empty, then $h_{c*} = k - 1$.

k -cliques in the worst case, as theoretically at least every subset of nodes with less than k nodes can be a potential k -clique. However, if the unknown graph is not extremely dense then most sets of nodes will not be potential k -cliques since a set of nodes is a potential k -clique only if it keeps the conditions defined in Corollary 1. Thus for small values of k , calculating $Clique^*$ is feasible.

Figure 1 presents an example of this heuristic. The size of the desired clique is 6. The next node that will be explored using $Clique^*$ is either H , I or J , as they are connected to a potential k -clique of size three (nodes S, E and F). Note that (S, E, F) is a potential k -clique because $|(S, E, F)| + |gen(S, E, F)| \geq 6$ (since $gen(S, E, F)$ is (H, I, J)).

Probabilistic Heuristic

In many domains the exact searched graph is unknown but there is some knowledge on the probability of the edges. For example, if the searched graph is the world wide web then it is well-known that it behaves like a scale free graph in which the degree distribution of the graph vertices follows a power law (i.e. the probability of a vertex having a degree x is x^β for some exponent β) (Bonato and Laurier 2004). Another example that is common in Robotics is a navigator robot in an environment represented by a graph. The robot may have an imperfect vision of the environment which could be represented as an error probability on the existence of an edge in the graph (Thrun, Burgard, and Fox 2005). In such cases, it seems that exploiting edge probabilities should significantly affect the choice of which vertex to explore next.

By adding probabilistic knowledge, the problem of searching for a k -clique in an unknown graph $G = (V, E)$ can be modeled as a finite-horizon Markov Decision Problem (MDP) (Puterman 1994) as follows: The *states* are all possible pair combinations of (G_{known}, V_{gen}) (of all possible graphs and sets of nodes). The *actions* are the exploration actions applied to a node in V_{gen} . The *transition function* between two states (old and new state) is affected by the existence probability of edges that were explored in the new state. Finally, the reward of every action is -1 (corresponding to the exploration cost of a node). The policy is to determine in each iteration which node to choose. An optimal policy minimizes the expected cost.

If an upper bound to the number of nodes in the unknown graph is known then it is theoretically possible to employ an MDP solver such as Value Iteration or Policy Iteration (Howard 1960) to solve the problem optimally. Unfortunately, the size of the MDP state space grows doubly exponentially with the number of vertices in the graph, as it contains all possible subgraphs of G . For example, a graph with 10 vertices requires $O(10!2^{45})$ MDP states: $10!$ for every possible order of exploring the 10 vertices, and 2^{45} for all the possible graphs with 10 vertices (an undirected graph with 10 vertices has at most $\frac{10 \cdot 9}{2} = 45$ edges). This prohibits techniques that require enumerating all of the states such as Value Iteration, Policy Iteration. A possible alternative is to use LAO* (Hansen and Zilberstein 2001), which finds the optimal solution without searching all the possible states. However it is known that LAO* usually requires

iterating over large parts of the state space and require significant overhead. We have implemented LAO* and found it to be inefficient, failing to produce an optimal policy in reasonable time for very small graphs (over 10 nodes).

RClique*

In order to still exploit the probabilistic knowledge, we propose a Monte Carlo based sampling technique that combines $Clique^*$ as a default heuristic and sampling of the state space. We call this heuristic *Randomized Clique** or in short *RClique**. This heuristic samples states from the MDP state space described above, using the probabilistic knowledge of the graph to generate the sampled states. Like $Clique^*$ and *KnownDegree*, *RClique** is invoked when choosing which vertex to explore next and chooses to explore the vertex with the lowest heuristic value. The *RClique** heuristic is designed to approximate the expected exploration cost based on samples of the MDP state space described above.

Algorithm 1 presents the pseudo code for *RClique**. The *RClique** heuristic requires the following two parameters : (1) *MaxDepth* specifies the maximum depth of every sample and (2) *NumOfSampling* specifies the number of samples to construct. In each sample, G'_{known} and G'_{gen} are initialized by G_{known} and G_{gen} (lines 3–4 in Algorithm 1). Then a vertex v' is chosen (from G'_{gen}) for simulated exploration using the $Clique^*$ heuristic (line 6). A simulated exploring action (*SimulatedExplore*) is then performed (line 7), removing v' from V'_{gen} and updating G'_{known} and V'_{gen} by adding an edge between v' and every other vertex in V'_{gen} with probability p , where p is the edge probability known to the agent. This process continues until either a k -clique has been found in G'_{known} or after *MaxDepth* iterations have been performed. If a k -clique has been found after d simulated exploration actions, the value stored for this sample is d (line 10). If the maximum depth has been reached but no k -clique was found, then the default h_{c*} is added to d , to provide a heuristic estimating of the remaining exploration cost (line 11). Finally, the average sample value is returned (line 13). This value represents an approximation of the expected exploration cost of choosing to explore vertex v . Note that the agent does not have any knowledge of vertices that are not in V_{known} . Therefore, since all the edges connected to vertices in V_{exp} are already known, the simulated exploration in *SimulatedExplore* adds edges only between vertices in V_{gen} .

Generating a random state (line 7) requires $O(|V_{gen}|)$ computational steps in the worst case - checking if the “explored” vertex is connected to every other generated vertex. Checking if the new state contains a k -clique requires $O(d^k)$ steps, where d is the degree of the “explored” node. This is repeated for a maximum of $MaxDepth \times NumOfSampling$ times. In the worst case, *RClique** is applied for every node in the graph (when no k -cliques exists). Therefore the total computational complexity of *RClique** is $O(|V| \cdot MaxDepth \cdot NumOfSampling \cdot d^k)$.

*RClique** has several features. It uses the probabilistic knowledge to generate the states during the sampling (line 7). It utilizes $Clique^*$ when choosing the target vertex during the sampling (line 6) and in estimating exploration cost

Algorithm 1: *RClique** cost function

Input: v , Explored vertex
Input: k , Size of desired clique
Input: $MaxDepth$, Max sample depth
Input: $NumOfSampling$, Number of samples
Output: The expected exploration cost

```
1  $r \leftarrow 0$ 
2 for  $i=1$  to  $NumOfSampling$  do
3    $G'_{known} \leftarrow G_{known}$ 
4    $V'_{gen} \leftarrow V_{gen}$ 
5   for  $d=1$  to  $MaxDepth$  do
6      $v' \leftarrow chooseNext(V'_{gen})$ 
7      $SimulatedExplore(v', G'_{known}, V'_{gen})$ 
8     if  $hasClique(k, G'_{known})$  then break
9   end
10   $r \leftarrow r + d$ 
11  if  $hasClique(k, G'_{known}) = False$  then  $r \leftarrow r + h_{c*}(v')$ 
12 end
13 return  $r / NumOfSampling$ 
```

in case the maximum depth has reached (line 11). In addition, it is an anytime algorithm: one can always add more samples in order to improve the average sample cost.

Experimental Results

We empirically compared the different heuristics by running simulations on various graphs. We varied (1) the size of the graphs, (2) the graph structure, (3) the initial state and (4) the desired clique size ($k = \{5, \dots, 9\}$). We compared our heuristics to a baseline algorithm, in which the agent chooses randomly which vertex to explore next. In addition, for every particular problem instance the optimal offline exploration cost was calculated (assuming full knowledge of the graph). The optimal cost is the shortest path from the initial vertex to the closest k -clique plus the exploration cost of exploring that clique ($k - 1$). Clearly no algorithm can do better than that, and thus it is a lower bound on the exploration cost of an optimal algorithm.

Simulated Graphs

We experimented with simple random graphs and scale free graphs. In random graphs the probability that an edge exists between any two vertices in the graph is constant. These graphs have been extensively used in many computer science problems as an analytical model and as benchmarks for evaluating algorithm efficiency. In scale free graphs the degree distribution of the vertices can be modeled by power laws, i.e. $p(degree(v) \geq x) = x^{-\beta}, \beta \geq 1$. Many networks, including social networks and the Internet exhibit a degree distribution that can be modeled by power laws. Since one of the domains which we are interested is the Internet, it seems natural to run experiments on this class of graphs as well. There are several scale free graph models. In this paper we focused on graphs generated by the graph model of Eppstein and Wang (2002) due to its simplicity.

In Figure 2 we compare the average total exploration cost on random graphs of the non-probabilistic heuristics,

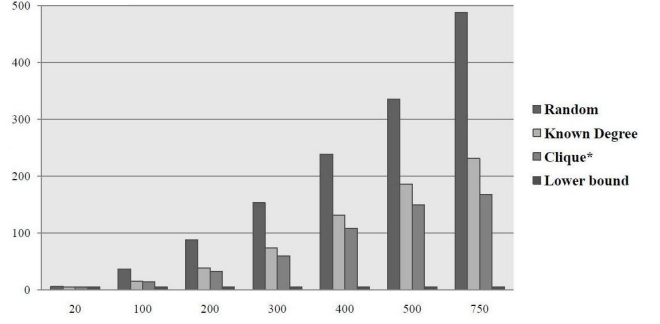


Figure 2: Non probabilistic heuristics on random graphs.

KnownDegree and *Clique**. We varied the vertex degree (8,12,16) and the searched k -clique (5,6). Each combination was simulated for 100 times on random graphs of various size. The x-axis displays the number of vertices in the graph and the y-axis displays the average of the exploration cost. Since similar results were obtained for all 6 combinations, our data points are the average of the 600 cases.

As can be seen, both heuristics proposed in this paper, *KnownDegree* and *Clique**, outperform the random approach, amounting to a reduction of more than 60%(!) of the total exploration cost. On the other hand, as the graph size grows the difference between the lower bound of the total exploration cost (computed by the optimal offline algorithm described above) and the total exploration cost of both heuristics increases. Another observation is that the *Clique** heuristic is more effective than *KnownDegree* for random graphs. This difference between the two heuristics increases as the number of vertices in the graph grows, achieving a reduction of 30% in random graphs with 750 vertices.

We also performed the same experiments on scale-free graphs. The results displayed two interesting phenomena. First, the reduction in the exploration cost of our heuristics over the random approach grows significantly to more than 80%. Second, the superiority of *Clique** over *KnownDegree* heuristic becomes insignificant (we used standard t -test). The reason for this result is that in scale free graphs, there are several "hub" vertices that have very high degree, and many vertices with very low degree (this creates the power law effect on the vertex degrees). Thus, exploring nodes with high degree is more likely to gain more knowledge of the graph.

Simulated Graphs with Probabilistic Knowledge

*RClique** assumes that probabilistic knowledge about the existence of an edge connecting two vertices in V_{gen} is available. To empirically evaluate *RClique** we simulated this probabilistic knowledge by a priori defining a probability for every possible edge between generated nodes, i.e., $\forall e = (v_1, v_2), v_1, v_2 \in V_{gen}, P(e \in E) = [0, 1]$ is known.

To determine different levels of uncertainty for an entire test configuration, we defined a variable *noise* that determines the *maximum* uncertainty of a single edge. For every possible edge e the agent is given $P(e \in E) = 1 - rand(0, noise)$ if $e \in E$ or $P(e \in E) = rand(0, noise)$

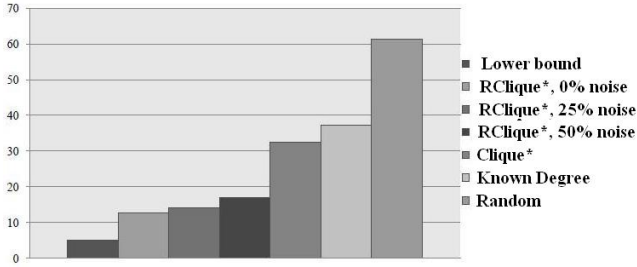


Figure 3: Random graphs, various levels of noise.

otherwise, where $rand(a, b)$ is a random number uniformly distributed between a and b . Clearly if $noise = 1$ the agent has random knowledge about the existence of edges in G .

We ran experiments for $RClique^*$ approach using 250 samples and a depth of 3 and various levels of noise. Figure 3 shows the average exploration cost of searching for a 5-clique in a graph of 100 vertices and 800 edges. The y-axis shows the average exploration cost. The bars denote the different heuristics, including (1) *KnownDegree*, (2) *Clique**, (3) $RClique^*$ with various levels of noise (0%, 25%, 50%), (4) random approach and (5) the lower bound provided by the optimal offline algorithm. As expected, an agent with less noise achieves a lower exploration cost. However, even with a setting of 50% noise $RClique^*$ outperforms *Clique** by a factor of almost 2. We have also evaluated the effect of different sampling depths on a $RClique^*$ heuristic using 250 samples and 50% noise. Depth 1,2,3,4 and 5 yielded an exploration cost that was 5.34, 4.15, 2.65, 2.52, 2.32 times the lowerbound respectively. Thus it seems that while deeper sampling reduces the exploration cost of the search, there is no substantial reduction beyond depth 3.

Real Unknown Graphs from the Web

We have built an online search engine designed to search for a k -clique in a real graph - the web. Specifically, we implemented a web crawler designed to search for a k -clique in academic papers available in *Google Scholar* web interface (denoted hereafter as GS). Each paper found in GS represents a node, and citations between papers represents the edges. We call the resulting graph the *citation web graph*. Naturally, the connection in context between papers is bidirectional (although two papers can never cross cite each other), thus we model the citation web graph as an undirected graph. The motivation behind finding cliques in the citation web graph is to find relevant significant papers discussing a given subject (as discussed in the introduction). This can be done by starting the clique search with a query of the desired subject.

The web crawler we implemented operated as follows. An initial query is sent to GS. The result is parsed and a list of hypertext links referencing academic work in GS is extracted. The crawler then selects which link to crawl to next, and follows that link. The resulting web page is similarly parsed for links. This process is repeated until a k -clique is found. First we ran the crawler with random walk, i.e., choose randomly which link to explore next. Figure 4 shows

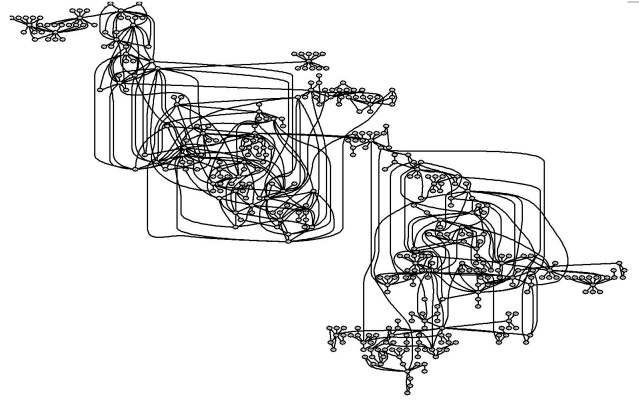


Figure 4: Citation web graph from a random walk in GS.

4-clique				
Algorithm	Cost	Runtime	Ratio	Found
<i>Random</i>	62.73	68.64	1.09	15
<i>KnownDegree</i>	54.73	28.59	0.52	14
<i>Clique*</i>	13.23	15.68	1.19	22

5-clique				
Algorithm	Cost	Runtime	Ratio	Found
<i>Random</i>	101	99.86	0.99	0
<i>KnownDegree</i>	82.91	44.59	0.54	6
<i>Clique*</i>	41.86	48.55	1.16	17

Table 1: Online search results of the GS web citation graph.

an example of the webgraph generated by a random crawl, starting with a GS query of "Sublinear Algorithms".

We have performed 25 such random walks, starting from different queries of 25 computer science topics, and generated 25 corresponding citation web graphs. As could be expected, the resulting graphs exhibit a power law distribution of the node degrees. Interestingly, many of the citation web graphs contained 4-cliques (95%) and 5-cliques (70%). On the other hand, very few (25%) contained larger cliques. Note that every random walk was halted after exploring several hundreds of nodes, and larger cliques may be found by further exploration.

Table 1 presents the results of 22 online GS web crawls, performed using each of the non-probabilistic heuristics and the random baseline. Each web crawl was halted after exploring 100 web sites. We used computer science related terms (e.g. "Subgraph-Isomorphism", "Online Algorithms") to start the crawl in GS. The *Cost* column represents the number of nodes explored until a k -clique was found and the *Runtime* column represents the runtime in seconds. If after exploring 100 web sites the desired clique had not been found, a cost of 101 was set. The *Ratio* column displays the average number of seconds required to explore a node (*Runtime* divided by *Cost*). The column *Found* represents the number of crawls in which the k -clique has been found.

Clearly *Clique** requires exploring significantly fewer nodes than both *KnownDegree* and *Random*. Much more 5-clique were found with *Clique** in comparison

with *KnownDegree* (17 vs. 6), exploring on average half of the nodes. However, due to the large overhead required in *Clique** the difference in runtime is not large (*KnownDegree* is even faster for 5-cliques). Performing a larger scale online experiment is beyond the scope of this work, as it requires leasing wide Internet bandwidth.

Consider the results in the *Ratio* column of Table 1, which are the the ratio between the search runtime and the number of explored nodes. For example, when searching for a 4-clique, the average number of seconds required to explore a node was 1.1 for *Random*, 0.52 for *KnownDegree* and 1.19 for *Clique**. The slow runtime per node exhibited by *Clique** can be explained by the additional runtime of running *Clique** in comparison with *KnownDegree* and *Random* which requires only $O(1)$ operations to choose the next node to explore. The difference in the runtime per node between *KnownDegree* and *Random* can be explained by the runtime required to search for a k -clique in the known part of the graph (G_{known}). As more nodes are explored, G_{known} grows, demanding more time to search for a k -clique. The fact that *KnownDegree* finds a k -clique by exploring less nodes than *Random* (54.73 as opposed to 62.73 for 4-clique), explains the larger runtime per node of *Random*. A crucial part of the search runtime is due to the time required to fetch a web page from the Internet. This introduces significant variability to the runtime results, as the time needed to fetch a web page depends on the location of the web page in the Internet and the state of the Internet route to it at the time that it is fetched. Thus it is not possible to analyze the runtime per node conclusively with a dataset that contains only 22 instances.

Conclusion and Future Work

In this paper we have analyzed the k -clique problem in unknown graphs. A best first search framework was proposed, with two non-probabilistic heuristics, *KnownDegree* and *Clique**. *RClique** heuristic has been presented that utilizes edge probability knowledge if available. Empirical evaluation has shown that these heuristics achieve substantial reduction in exploration cost.

There are many open questions and future directions. We intend to perform a large scale online crawling using the proposed algorithms, combining the clique search with a content based textual data mining. Moreover, this paper has focused on a special pattern - a clique. We intend to generalize the approaches presented in this paper to solve the general subgraph isomorphism problem in unknown graphs. Additionally, we intend to research more complex exploration cost model, such as a physical based model in which exploring a node requires an agent to move to its geographic location. Multiple agents collaborating in unknown graphs is another interesting future direction.

Acknowledgments

This research was supported by the Israeli Science Foundation (ISF) grants No. 728/06 and 305/09

References

- Altshuler, Y.; Matsliah, A.; and Felner, A. 2005. On the complexity of physical problems and a swarm algorithm for the k -clique search in physical graphs. *ECCS*.
- Bollobas, B., and Erdos, P. 1976. Cliques in random graphs. In *Math. Proc. Camb. Phil. Soc.*, 419–427. Great Britain: MPCPS.
- Bonato, A., and Laurier, W. 2004. A survey of models of the web graph. In *CAAN*, 159–172.
- Bron, C., and Kerbosch, J. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* 16(9):575–577.
- Eppstein, D., and Wang, J. 2002. A steady state model for graph power laws. In *2nd International Workshop on Web Dynamics*.
- Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artif. Intell.* 129(1-2):35–62.
- Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press.
- Kalyanasundaram, B., and Pruhs, K. R. 1994. Constructing competitive tours from local information. *Theoretical Computer Science* 130(1):125–138.
- Pullan, W., and Hoos, H. H. 2006. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research* 25:159–185.
- Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience.
- Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.
- Tomita, E., and Kameda, T. 2007. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. of Global Optimization* 37(1):95–111.