NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Analysis of prefetching methods from a graph-theoretical perspective

Alexandros Panagiotis - P - Perikleous

Supervisor: **Vassilios Zissimopoulos,** Professor

ATHENS

JULY 2016

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάλυση μεθόδων προανάκτησης μέσα από ένα γραφοθεωρητικό πρίσμα

Αλέξανδρος Παναγιώτης - Π - Περικλέους

Επιβλέπων: **Βασίλειος Ζησιμόπουλος,** Καθηγητής

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2016

BSc THESIS

Analysis of prefetching methods from a graph-theoretical perspective

Alexandros Panagiotis - P - Perikleous

S.N.: 1115201100119

SUPERVISOR: **Vassilios Zissimopoulos,** Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάλυση μεθόδων προανάκτησης μέσα από ένα γραφοθεωρητικό πρίσμα

Αλέξανδρος Παναγιώτης - Π - Περικλέους

Α.Μ.: 1115201100119

ΕΠΙΒΛΕΠΩΝ: **Βασίλειος Ζησιμόπουλος,** Καθηγητής

ABSTRACT

It is important to highlight the role Content Distribution Networks (CDNs) play in rapidly growing Internet topologies. They are responsible for serving the lion's share of Internet content to the end users by replicating it from the origin server and placing it to a caching server closer to them. Probably the biggest issues CDNs have to deal with revolve around deciding which content gets prefetched, in which surrogate/caching server it is placed and allocating storage to each server in an efficient manner. We will focus on the content selection/prefetching problem extending the work done by Sidiropoulos et al. (World Wide Web Journal, vol. 11, 2008, pp. 39-70). Specifically, we are trying to determine how their clustering algorithm can work in specific environments in comparison with an approach used to solve the surveillance game in graphs as discussed by Fomin et al (Proc. 6th Int'l Conf. on FUN with Algorithms, 2012, pp.166-176) and Giroire et al. (Journal of Theoretical Computer Science, vol. 584, 2015, pp.131-143). Along the way, we provide another definition for cluster cohesion that accounts for edge cases. Finally, we define an original problem, which consists of partitioning a graph into a predefined amount of disjoint clusters of optimal average cohesion.

SUBJECT AREA: Algorithmic Operations Research

KEYWORDS: surveillance number, densest subgraphs, Content Distribution Networks, graph partitioning, cluster cohesion

ΠΕΡΙΛΗΨΗ

Είναι σημαντικό να τονίσουμε το ρόλο που τα Δίκτυα Διανομής Περιεχομένου (CDNs) παίζουν στις ταχέως αναπτυσσόμενες τοπολογίες του Διαδικτύου. Είναι υπεύθυνα για την εξυπηρέτηση της πλειοψηφίας του περιεχομένου του Διαδικτύου στους τελικούς χρήστες αντιγράφοντας το από το διακομιστή προέλευσης και τοποθετώντας το σε έναν διακομιστή πιο κοντά τους. Τα μεγαλύτερα ίσως προβλήματα που αντιμετωπίζουν τα CDNs έχουν να κάνουν με την επιλογή του περιεχομένου που πρέπει να προανακτηθεί αλλά και την επιλογή ενός κατάλληλου διακομιστή μεσολάβησης στον οποίο θα τοποθετηθεί. Εμείς θα επικεντρωθούμε στο πρόβλημα προανάκτησής περιεχομένου επεκτείνοντας την έρευνα που έγινε από τον Σιδηρόπουλο κ.α. (World Wide Web Journal, vol. 11, 2008, pp. 39-70). Συγκεκριμένα, θα προσπαθήσουμε να αποφανθούμε πώς η μέθοδος συσταδοποίησής τους μπορεί να δουλέψει σε συγκεκριμένα περιβάλλοντα σε σύγκριση με μια άλλη προσέγγιση που χρησιμοποιείται για την επίλυση του παιχνιδιού επιτήρησης σε γράφους όπως διερευνήθηκε από τον Fomin κ.α. (Proc. 6th Int'l Conf. on FUN with Algorithms, 2012, pp.166-176) και τον Giroire κ.α. . (Journal of Theoretical Computer Science, vol. 584, 2015, pp.131-143). Στην πορεία, δίνουμε και έναν άλλο ορισμό για τη συνοχή των συστάδων που καλύπτει και οριακές περιπτώσεις. Τέλος, ορίζουμε ένα καινούριο πρόβλημα, τη διαμέριση δηλαδή του γράφου σε έναν προκαθορισμένο αριθμό ανεξάρτητων συστάδων με βέλτιστη μέση συνοχή.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αλγοριθμική Επιχειρησιακή Έρευνα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: π.χ. αριθμός επιτήρησης, πυκνότεροι υπογράφοι, δίκτυα διανομής περιεχομένου, διαμέριση γράφων, συνοχή συστάδας

This thesis is dedicated to my brother and parents for the support and encouragement they showed through this process.

ACKNOWLEDGMENTS

.

I would like to thank supervisor professor Vassilios Zissimopoulos for cooperating with me and helping me see this thesis through. This project would not have been possible without his support.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

First and foremost, it is plainly obvious that the Web is growing by the minute. Since 1990 when the first web page was born, websites have increased dramatically in size through the addition of videos, images, fonts, stylesheets, scripts etc. In particular, in 2010 the average web page size was 702 kb, while in 2016 it is 2232 kb. As a result, the demands being placed on the Internet bandwidth nowadays have more than tripled. A naive solution to this issue would be a bandwidth increase, but such a decision comes with great financial cost. More to the point, this would be just a band-aid to a far more complex issue, since users would in time learn to use the added bandwidth in a greedy fashion thus clogging up the network once again. If we take into consideration the difficulty to make any changes in the last mile infrastructure, that has basically been unchanged for a century, then this approach is rendered infeasible.

Instead of changing the infrastructure, researchers focused mainly on improving content delivery itself. One of the methods used to achieve this was caching. This however had its own drawbacks and so traditional caching was combined with a variety of other techniques like prefetching, consistency maintenance and cooperative caching to make content distribution quicker and more efficient. This was partly achieved, even though each method above has well-documented issues as well [7].

These issues are exactly what CDNs promise to resolve. The fact that the CDN acts as an intermediary between the origin server and the end users is not immediately apparent to them, since they all access the website by typing it in their browser. Content providers such as media companies and e-commerce vendors pay CDN operators to deliver their content to their audience. In turn, a CDN pays ISPs (Internet Service Providers), carriers, and network operators for hosting its surrogate servers from all over the world in their data centres. Besides better performance and availability, CDNs also offload the traffic served directly from the content provider's origin infrastructure, resulting in possible cost savings for the content provider.[1] In addition, CDNs provide a degree of protection from DoS (Denial of Service) attacks by using their large distributed server infrastructure to absorb the attack traffic.

How exactly does it work then? When the content provider's server receives the request, it only serves the bare minimum in terms of content. Anything more than the page skeleton and some basic graphics is forwarded to the CDN provider, that then decides using its own proprietary algorithms which caching server will serve which client based on a number of variables [5]. This may be measured by choosing locations that are closest to the clients geographically, are the least expensive, have the least traffic both currently and historically or any combination of the above. Whichever server is selected though, upon first request it will not yet have the objects cached and will download them from the content provider server. Upon subsequent requests the objects will be served by the selected caching server. This process is shown in Figure 1. The IP address of the user making the request is retrieved and the CDN provider’s (in this case Akamai’s) proprietary algorithms use that information to select the most appropriate edge server to serve the content to the user. Domain Name System (DNS) redirection is also being used to offload traffic from busy servers. In the following subsection we will examine how CDNs got to where they currently are with a brief overview of their progression through the years and project their next major evolutionary step.

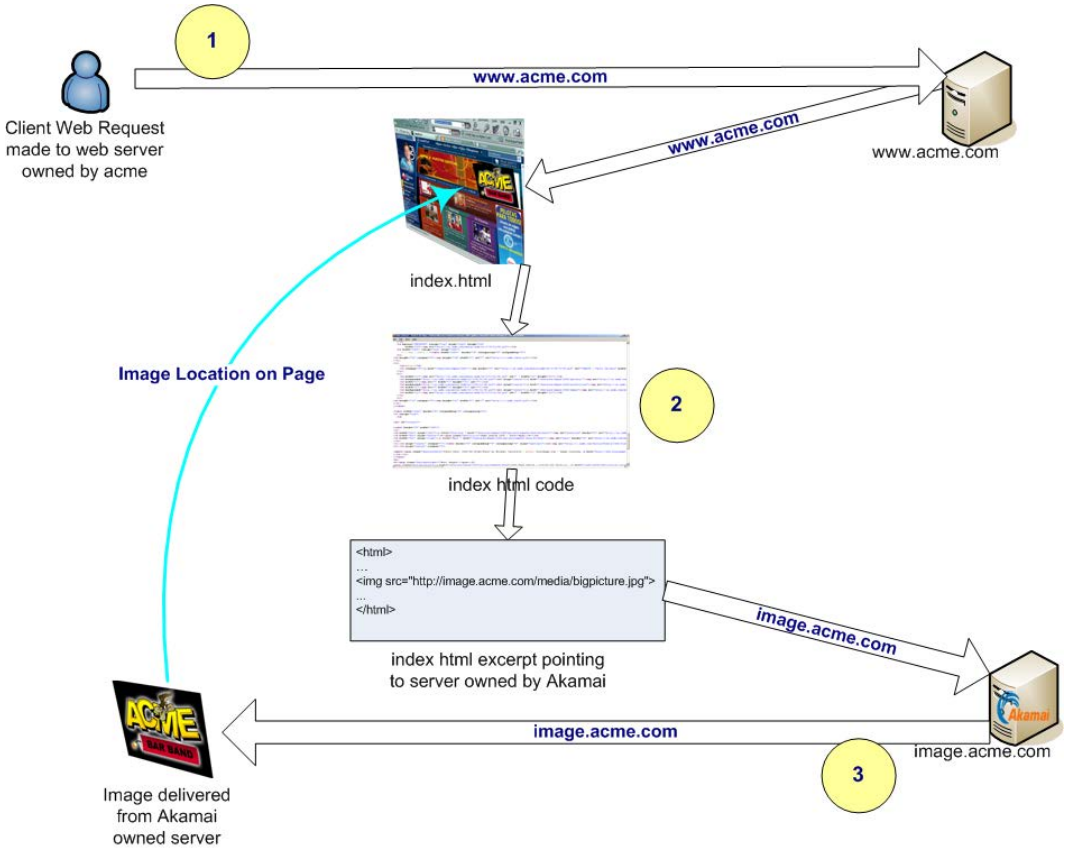

Figure 1: Content delivery process at a leading CDN provider, Akamai technologies [6]

1.1 CDN Timeline

1.1.1 Brief History of CDNs

CDNs first came about to address the need for more bandwidth and a growing volume of content. Changes being made in server deployment with increasing adaptation and gradual improvements of caching techniques helped give rise to CDNs.

At first, CDNs concentrated mainly on delivering static and dynamic content to the users. Intelligent routing, distributed replication, efficient prefetching schemes, storage capacity allocation and object placement strategies were atop the list of priorities for the first generation of Content Delivery Networks. The flash crowd problem discussed in more detail here [17] [18] was addressed during that first stage in the evolution of CDNs. One noteworthy fact is that Akamai Technologies, one of the leading CDN providers of our time started out as an MIT research effort to combat that specific problem.

The next step for CDNs was to focus on Video-on-Demand, audio and video streaming and increasing interactivity with the users. This is the stage in the CDN timeline that we first saw peer to peer production, cloud computing and energy awareness for content delivery and maintenance. Unfortunately, most of these techniques are still in the developmental phase and have not been used extensively just yet.

Recently, the domination of mobile phones and Voice-over-Internet among many other societal and technological changes have forced telecommunications service providers

into a corner. Traditional revenue streams have taken a big blow and surviving in such a harsh economic climate required some out of the box thinking. Telcos accepted this challenge and came up with their own version of CDNs, one that provides several advantages over regular CDNs. For starters, they already own the network infrastructure and are connected to the end users themselves. This allows them to save up on costs and offer a superior experience to the user. This is just the tip of the iceberg however [10] [11].

1.1.2 Future of CDNs

What is the next step for Content Delivery Networks then? Well for one thing they are expected to be driven entirely by the users themselves. In essence, this means that the content delivery will be autonomous, self-sustainable and completely adaptable. We are still a long way off from this being the norm however.

One thing is for certain: CDNs are not going anywhere for the foreseeable future. In fact, several reports project that the CDN market in the US will triple in revenue by 2019 [8] [9]. Nowadays, improving CDN performance is key for any online business that wants to drive sales and attract customers through websites. A study conducted by Forrester Consulting on behalf of Akamai technologies has shown that users do not wait for more than 2 seconds for content to load before 47 percent of them leave the website. More to the point, customer loyalty is closely tied to how quickly a Web site loads. In fact, 61 percent of regular online shoppers insist on pages loading quickly in order to buy a product from that particular retailer [7]. Therefore, it is of utmost importance for businesses to invest a great deal of effort and financial resources in optimizing content delivery.

1.2 Our contribution

How exactly is all this connected with our work? Well for CDNs to succeed at reducing the load on the origin server and decreasing overall latency they replicate the content closer to the edges of the Internet. Doing so however requires an efficient prefetching policy that can predict and cluster together the most likely websites to be visited next by the user. This has been the subject of extensive research both from a practical viewpoint [1] and a graph-theoretical one [2][3]. In this paper, we will dissect the latter approach and try to determine whether it can be more efficient than the former one in certain cases. We will end that introduction with some preliminary notions regarding graphs that will be put to use later in our work.

1.3 Preliminaries on graphs

A *graph* G is defined as a pair of two sets: the set of vertices $V(G)$ and the set of edges $E(G)$. A *vertex* is also called a node, while an *edge* is an unordered pair of two nodes. The set of edges is therefore a subset of the set of all possible unordered pairs of vertices.

A *directed graph* (or digraph) G is similar to an undirected graph with the variation that the edge set $E(G)$ is a subset of the set of all possible ordered pairs of vertices. The vertices connected by an edge are called *neighbouring* vertices. The open *neighbourhood* of a node $v \in V(G)$ is defined as $N(v) = \{u \in V(G) : [v,u] \in E(G)\}$ while the closed neighbourhood is defined as $N[v] = N(v) \cup \{v\}$. The *degree* of a node $v \in V(G)$ is

defined as $d(v) = |N(v)|$. The minimum degree of G is denoted by $\delta(G)$, where $\delta(G) = min_{v \in V(G)} d(v)$. The maximum degree of G is denoted by $\Delta(G)$, where $\Delta(G) = max_{v \in V(G)} d(v)$.

A connected graph that contains no cycles is called a *tree*. For every non-root node there is a *father* and every node that lies one level below a certain node is called *child* of that node. A *leaf* is a tree node of degree 1. The *depth* of the tree is the maximum distance from a leaf to the root.

2. CLUSTER COHESION

Before we examine both approaches used to solve the prefetching problem in terms of efficiency, we want to take a closer look at clusters themselves and the importance of interconnectedness within each cluster. To do this, we are going to borrow the definition of community cohesion from [1].

Naturally, it is in our best interest to look for and identify communities that are strongly linked internally. When a user enters such a cohesive cluster, it is far more likely that he will stay in it because the pages are so densely linked that he will be almost forced to select a link that directs him to another page in the same collection. Also the user's interests probably coincide with the subject matter of the collection, making it even more difficult for the user to leave it and jump to another one.

Considering the importance cluster strength or cohesiveness play in clustering methodologies and prefetching by extension, it is imperative that we use a bulletproof definition for it. In [1] they used:

$$c(x) = \frac{number\ of\ 'external' links}{number\ of\ 'internal' links} \quad (1)$$

and they wanted to minimise this factor for all the clusters:

$$F(G, C) = \frac{1}{|C|} \sum_{\forall x \in C} c(x) \quad (2)$$

where G is the graph, C is the set of clusters and $|C|$ is the cardinality of C.

We wanted to improve upon (1) by accounting for singleton clusters as well. At the beginning of the clustering process each cluster only has one node in it and that is the kernel node. At that point, this cluster has no internal links and zero or more external links. This is where (1) falters, because $\frac{a}{0}, a \geq 0$ is undefined. To remedy this, we could add 1 to the denominator like so:

$$c(x) = \frac{number\ of\ 'external' links}{number\ of\ 'internal' links + 1} \quad (3)$$

This has little to no effect on the clustering process. However, the formula is now bulletproof from edge cases that could cause cluster cohesion to veer towards infinity.

3. COMPARING THE SURVEILLANCE GAME APPROACH AND C3I AT CLUSTERING

3.1 C3i algorithm

The correlation clustering communities identification algorithm (in short C3i) was created by Sidiropoulos et al. in an effort to identify all the communities/clusters in the Web graph. The community as defined in their work [1] is a subgraph V of a Web graph G with the feature that the sum of all degrees within the subgraph V is larger than the sum of all degrees towards the rest of the graph G. C3i is trying to optimize cluster cohesion at the maximum number of clusters starting from some initial nodes, called *kernel* nodes and will only stop expanding the clusters once all nodes belong to one. The algorithm is presented in detail in the following figures. (Figure 2)(Figure 3)(Figure 4)

Algorithm 1: C3i algorithm

```
Input: G(V,E) : Web site graph
K = 0;//set of clusters
repeat
Phase 1 : Selection of kernel nodes
        select_kernel_nodes(G);
        make each kernel node a cluster C_i;
        K = K ∪ C_i;
Phase 2 : Clustering procedure
        Cohesion_clustering(K,G);
until K cannot grow anymore
return K;
```

Figure 2: C3i algorithm

As we can observe from Figure 2, C3I consists of two phases. At first, it selects the kernel nodes of the clusters and creates a set of clusters based on this selection. Each of these clusters will initially just have one node, its kernel. After this phase is over, the clusters are expanded until all nodes belong to one. It should be noted that singleton clusters are valid with this approach.

We dissect this method phase by phase in the two following figures. In Figure 3 it is shown that before we can make a decision upon which nodes to mark as kernels we must first estimate their number. In [1] they use $n = \sqrt{|G|}$ as an approximation for this number and that is the formula we are going to use as well. Then, a combination of four methods is used to deduce which nodes are best suited to be kernels (Randomly, Degree-based, Pagerank-based, HITS-based).

```
select_kernel_nodes(G)
Input: G(V,E) : Web site graph
n=determine_kernel_number(G);
node_array[n];
r=sort_nodes_random(G,n);
d=sort_nodes_degree(G,n);
p=sort_nodes_PageRank(G,n);
h=sort_nodes_HITS(G,n);
for i from 1 to n/4
        node_array[i]=r[i];
i = n/4 +1;
//At the end of this loop i must be equal to n/2
j = 1;
count = 0;//counts the number of nodes we have
copied from each array to the node array.At the
end of each loop this must be equal to n/4
while i ≤ n/2
  if (checksimilarity(node_array,d[j])==0)
  //this node has not been selected yet
        node_array[i]=d[j];
        count++;
        i++;
        j++;
  else if (checksimilarity(node_array,d[j])==1)
        j++;
  end if
end while
i = n/2 +1;
j = 1;
count = 0;
while i ≤ 3n/4
  if (checksimilarity(node_array,p[j])==0)
    node_array[i]=p[j];
    count++;
    i++;
    j++;
  else if (checksimilarity(node_array,p[j])==1)
    j++;
  end if
end while
i = 3n/4 +1;
j = 1;
count = 0;
while i ≤ n
  if (checksimilarity(node_array,h[j])==0)
    node_array[i]=h[j];
    count++;
    i++;
    j++;
  else if (checksimilarity(node_array,h[j])==1)
    j++;
  end if
end while
```

Figure 3:C3i Phase 1

In Figure 4 Phase 2 is depicted. After creating the initial clusters, C3i will try to add neighbouring nodes to them or remove them in later steps always trying to minimize the

average cohesion for all the clusters (Recall that $F(G,C) = \frac{1}{|C|}\sum_{\forall x \in C} c(x)$) . Using the subroutine find_unions C3i will also look for possible cluster unions if the clusters' intersection is at least half the larger cluster and such a choice will lead to a reduction in the average cluster cohesion.

```
Cohesion_clustering(K,G)
Input: G(V,E) : Web site graph
Input: K : Set of clusters
Step 1:
repeat
        for each cluster c connected to K do
                Check_1_node_expand(c);
        end for
        find_unions(K);
until no change in K;
Step 2:
repeat
        for each cluster c in K do
                Check_1_node_remove(c);
        end for
until no change in K;
Step 3:
repeat
        for each cluster c in K do
                Check_x_nodes_expand(c,t);
        end for
        find_unions(K);
until no change in K;

Check_1_node_expand(c)
Input: c: cluster
for each node n connected to c
  if F(c) > F(c∪n)
    c = c ∪ {n}
  end if
end for

Check_1_node_remove(c)
Input: c: cluster
for each node n in c
  if F(c) > F(c n)
    c = c \ {n}
  end if
end for

Check_x_nodes_expand(c,t)
Input: c: cluster, t: integer
for each subgraph S(|S| ≤ t connected to c)
  if F(c) > F(c ∪ {S})
    c = c ∪ {S}
  end if
end for
```

```
find_unions(K)
Input: K : Set of clusters
For each cluster C_i in K from i=1 to n
 For each cluster C_j in K from j=i+1 to n
  if (|C_i ∩ C_j| ≥ max(|C_i|,|C_j|)/2
   if F(C_i) > F(C_i ∪ C_j)
 //the merging takes place iff the resulting
 clustering is better than the previous one
        C_i = C_i ∪ C_j
  end if
 end if
 end for
end for
```

Figure 4: C3i Phase 2

3.2 Surveillance game algorithm

The surveillance game basically boils down to this: The Web is modelled as a huge digraph G. Its nodes represent Web pages and its arcs represent hyperlinks among these pages. If we assume that a surfer begins from a starting node $v_0 \in V$, then our job is to make sure that whichever path the surfer chooses to traverse, he will never find himself asking for a non-prefetched object. That can only happen by determining the surveillance number of G starting in v_0, which is equal to the least number of Web pages prefetched at each step that avoid the Web surfer to wait. It is important to prefetch as few objects as necessary because of the network's bandwidth limitations.

This approach is demonstrated in Figure 5 with a very simple example. Assume that the surfer starts traversing this Web graph from vertex v_0. The neighbouring vertices marked in blue are v_1, v_2, v_3. Each vertex corresponds to a Web object (page). The goal of the observer (equivalently the CDN provider) is to mark and thus fetch the neighbouring objects before the surfer has a chance to reach them. The combination of the marked objects and the current object of the surfer forms a new cluster. This process is repeated every time the surfer moves to a new vertex. By making use of the surveillance number (equivalently the amount of resources the CDN provider has at its disposal) the observer can mark specific nodes further ahead and ensure that the surfer will always be satisfied.

We must note here that we consider the web browser to be all-knowing and able to see the entire graph. That is unrealistic as discussed in [3]. However, this approach will suffice for the purpose of our work.

A high-level description of this algorithm is shown in Figure 6 and a more detailed one is shown in Figure 7.

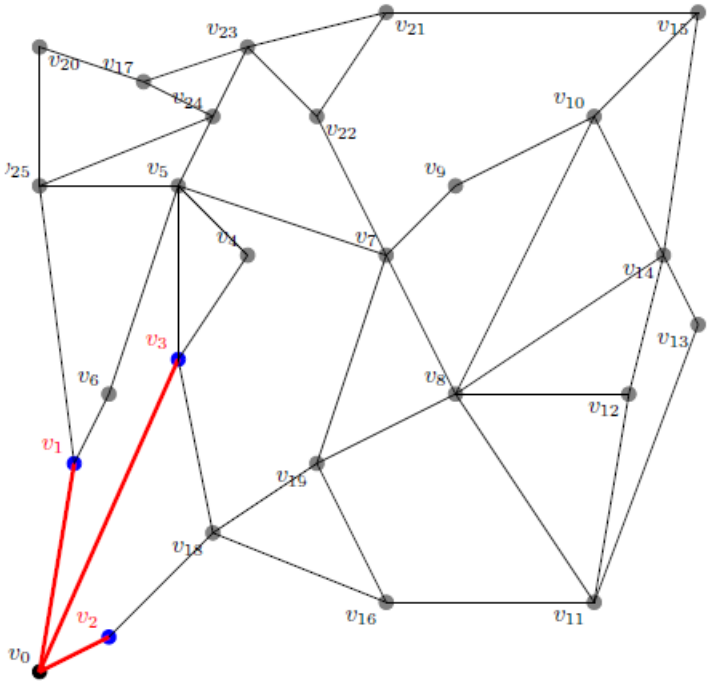

Figure 5: Prefetching example in a web graph assuming surfer starts from v_0

Algorithm 2: Surveillance game algorithm
Input : G(V,E) : Web site graph
Input : initial node v_0
Step 1: Find surveillance number
repeat
Step 2: Mark neighbors
Step 3: Use remaining marks(if any) to tag specific nodes ahead;
Step 4: Make cluster including marked nodes and current node until graph has no unexplored nodes

Figure 6: High-level description of the Surveillance game algorithm

```
Input : G(V,E) : Web site graph
Input : initial node v0
Snode = v0;//surfers current node
K = 0;//set of clusters
Marked = 0;//set of marked objects
Frontier = 0; //neighboring vertices
Marked = Marked ∪ {v0}
Frontier = find_unmarked_neighbors(Marked,v0);
sn=find_surveillance_number(G);
//Finding the surveillance number is trivial
for trees but very difficult for some graphs
mark_neighbors;
use remaining marks(if any) to tag specific
nodes ahead;
C = make_cluster(Marked,Snode);
K = K ∪ C;
while graph_still_has_unexplored_nodes{
//we detect where the surfer moves, mark the node
where he is in and create a new cluster that
includes that node and its neighbours
  Snode = scan_movement(G,Snode);
  Marked = Marked ∪ {Snode};
  Frontier=find_unmarked_neighbors(Marked,Snode);
  mark_neighbors;
  use remaining marks(if any) to tag specific
  nodes ahead;
  C = make_cluster(Marked,Snode);
  K = K ∪ C;
}
```

Figure 7: Surveillance game algorithm in detail

3.3 Differences between the two methods

The main differences between this approach and the one discussed in [1] is that we need to know the starting position of the surfer in order to deduce an efficient prefetching policy. In comparison, we do not need to know anything about the surfer before we begin clustering in the Web graph as in [1]. This algorithm only takes interobject relationships and specifically the existence of hyperlinks between objects into consideration. Logical correlation between objects and the direction of the links are of little concern for this algorithm. Furthermore, using the surveillance game approach we have to prefetch as many objects as is the surveillance number of the graph. In other words, if at the current position of the surfer the amount of marks we have (equivalently the resources we have to fetch a specific number of Web pages) trumps the amount of neighbours of the surfer, then we must make use of the remaining marks (or resources) to fetch objects that might be further ahead in the graph topology and form our cluster. The surveillance game approach does not forbid us from seeing further ahead or marking nodes that are not neighbours to any already marked nodes. These restrictions only hold true for the online and connected variants, respectively. Obviously, determining the surveillance number of

a graph and knowing exactly which nodes to mark at each step of the surfer is no trivial matter and the research on this is extensive[12][13].

Inside the loop there is always a comparison going on between the surveillance number and the number of neighbouring objects that could possibly be visited by the surfer. Because finding the surveillance number in some types of graphs is extremely difficult, we have decided to use an upper bound for it instead. The surveillance number of a graph is equal to its maximum degree in a worst-case scenario (and the starting node degree at best). This means that at every step we have resources equal to the maximum degree of the graph. If at a certain node we have unused marks (or resources) we assign them where they are most needed, at specific nodes ahead.

Figure 8 illustrates this method. The red nodes are the ones visited by the surfer. The blue nodes are the ones identified by our algorithm as neighbours. Assuming the surveillance number is 5, then the algorithm has to form a cluster with objects v_1, v_2, v_3 and of course the initial object v_0. Because we still have two marks remaining, these need to be assigned further ahead. For example, nodes v_7, v_8 could be fetched and added to the initial cluster. At step 2, when the user has visited object v_3 he must fetch 4 more neighbouring objects: v_4, v_5, v_{18}, v_{19} and one distant object: v_{14}. At step 3, when the surfer has visited v_{19} we will have to tag 5 unmarked objects. This is made possible by marking/tagging as many objects as the surveillance number of the graph (=5) and having marked the others in previous steps.

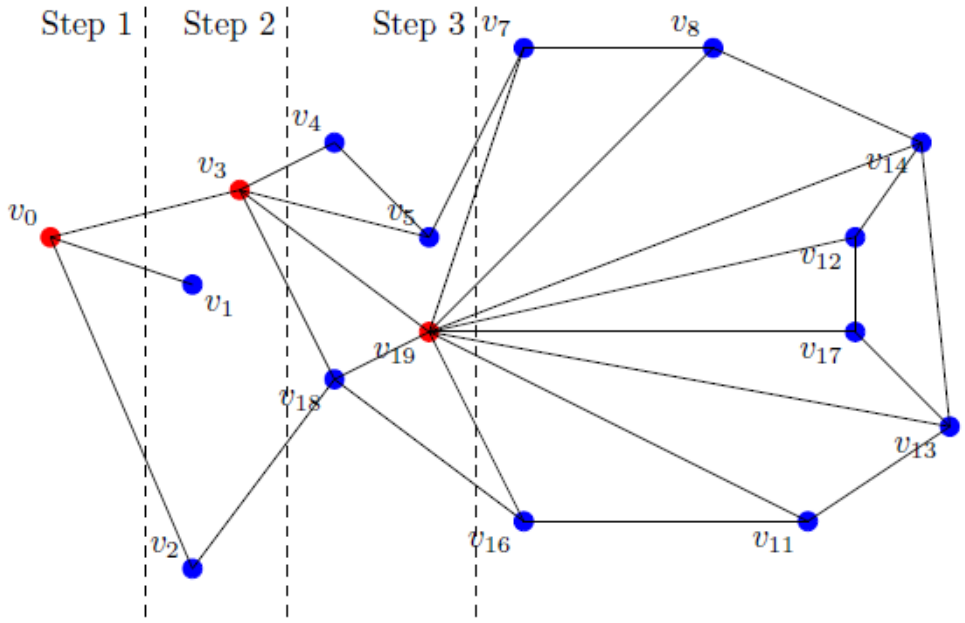

Figure 8: Surveillance game algorithm in a graph (Surveillance number = 5)

3.4 Similarities between the two methods

Having said all of that, a fair argument could be made that the surveillance game approach actually converges with the C3i method in specific problems and is indeed a relaxation of the latter. This is made clearer if we consider:

- We only use one node as kernel in each step (surfer's current position) instead of multiple kernel nodes.
- Complete knowledge of the Web graph is needed in both cases.

- We do not need to subtract nodes or unite clusters, as cluster formation is very specific as long as we are aware of the surfer's current position.

We will try to take this theory one step further and examine how these algorithms work in the case of trees in the next section.

4. APPLICATIONS IN TREES

It has been proven in [2] that the surveillance number of a tree rooted in a specific node v_0 can be computed in polynomial time $O(nlogn)$.

The algorithm given is recursive and is presented below. Finally, it is proven that for any tree T rooted in v_0, $sn(T, v_0) = max \lceil \frac{|N[S]|-1}{|S|} \rceil$ where the maximum is taken over all subtrees S of T containing v_0. The closed neighborhood $N[S]$ is the union of the subtree S and the subset of vertices in $V \setminus S$ having a neighbour in S:

$N[S] = N(S) \cup S.$

Let k $\geq$ 0. We define the function
$f_k : V(T) \to N$ in the following recursive way:
1) $f_k(v) = 0$ for any leaf v of T;
2) for any v $\in$ V(T) with d children,
$f_k(v) = \max\{0, d + \sum_{w \in C} f_k(w) - k\}$
where C is the set of children of v.

Figure 9: Computing the surveillance number in trees [2]

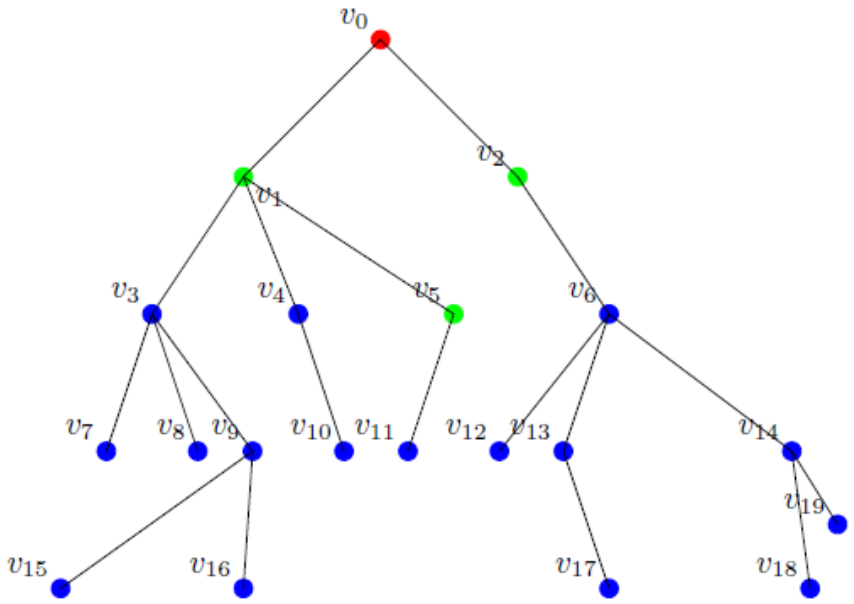

Figure 10: Surveillance game approach in a tree

This basically means that the surveillance number can be computed very quickly in regards to the size of the tree and as a result this approach is very efficient for creating clusters of objects in those environments (Figure 9). Look at the tree above for instance. Assume the surfer starts at v_0 and the surveillance number is 3. The green nodes are the ones marked by the observer for fetching and the red node is the position of the surfer. Combining the red and green nodes at each step will produce the needed clusters. The algorithm will tag neighbouring nodes at first and assign any remaining marks to specific nodes ahead. (Figure 10)

Is the C3i algorithm however equally as efficient in such cases? Unfortunately it is not and this can be proven with a very simple example.

Assume a basic tree (Figure 11). If we were to use the C3i method on this tree, it would start by looking for kernel nodes using a combination of randomness, HITS, PageRank and max degree criteria as we described. Let's consider for the sake of this example that three kernel nodes are used (v_0, v_3, v_6) and therefore three clusters are created and looking to expand. At first the cohesion of all these clusters will be equal to the number of edges touching the kernel nodes (3). These clusters will expand as shown.

Let's take cluster α for instance. Starting from v_3, it will have cluster cohesion equal to $c(x) = \frac{4}{1}$. After that, it may choose to expand adding both children v_7 and v_8. Then the cluster cohesion would further decrease down to $\frac{3}{2}$ and $\frac{2}{3}$ respectively. It cannot grow any further however because adding either v_9 or v_1 would yield higher cohesion. This leads to a situation where the surfer that currently is on node v_3 will not be able to visit all of the children objects equally quickly (v_9 in particular).

On the flip side, if cluster α chose to expand adding node v_1 either on the first or second step, it could do so. This can happen, because the cohesion would decrease

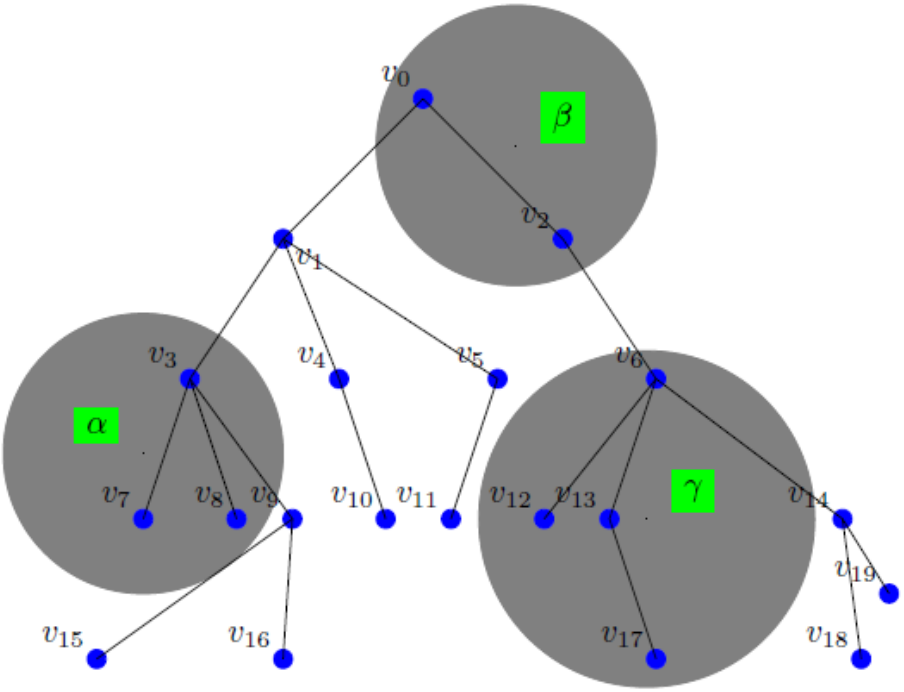

Figure 11: C3i in a tree

from $\frac{4}{1}$ to $\frac{6}{2}$ (adding v_1 on the first step). Following this, the cluster can grow to include the entire left subtree starting from root v_0, v_0 itself and v_2. It would get stuck at v_6, because at that point the cohesion would be $\frac{1}{12}$ and would worsen at $\frac{2}{13}$. The most likely course of action for cluster α however would be to start by adding its children in the cluster because it would lead to a more drastic drop in its cohesion. The next step would be to look for more kernel nodes out of the ones not in a cluster currently and check for possible unions.

It is very easy to notice the trend at play here. As long as their cohesion exceeds 1, clusters can have multiple opportunities for growing as previously shown. If it is below 1, expansion becomes very strict in the case of trees and amounts to picking amongst leaves that are neighbours to the cluster. If there are no such vertices nearby, the expansion stops.

This just goes to show that the tree partitioning of k disjoint clusters is not trivial. Even, in the case where $k = 2$, the final clustering can look dramatically different due to several key factors. For example, the choice of nodes to be used as kernels for our clusters is such a factor. If leaves assumed the role of kernel nodes, then that would lead to a very bad clustering. Another less obvious variable is the strategy we choose to follow for cluster expansion. If it is aggressive, meaning that we look for faster drops to the overall cohesion, then the clusters would very quickly arrive at a deadend after adding a few leaf nodes, because they cannot afford adding other nodes that would increase their cohesion. (e.g. Cluster α in Figure 10) In comparison, if the strategy chosen was more lenient, then the clusters would look to add nodes with higher degrees more frequently and thus it would take longer for them to get stuck. Clustering therefore can be affected by those aforementioned key choices.

At this point it would also be a good idea to take a step back and assess the quality of the clusters we have created. First of all, the linkage is very sparse. As expected, the vertices surpass the edges within each cluster by one. This does not make for very cohesive communities of objects. Unfortunately, this will always be the case in trees ($Total\ edges\ =\ Total\ vertices\ -\ 1$) and therefore their subtrees. Also, a surfer that visits node v_3 for instance and wants to visit node v_9 will not be able to do so quickly because it will not be fetched along the nodes (v_7, v_8). Having said that, while there is a chance in the third step of the algorithm that another cluster containing node v_3 will unite with this one there is no such guarantee, because it will have to run into the same problem at some point during its expansion and may not even reach the point of union.

The root of the issue lies in the fact that we emphasize cluster cohesion more than we should. Creating cohesive clusters is nowhere near as significant in trees as it is in graphs due to their unique topologies. .

5. GRAPH PARTITIONING

A worthwhile problem to look into is whether or not there can be a perfect graph partitioning. More formally, given an undirected graph $G = (V, E)$, where V denotes the set of n vertices and E the set of edges, can G be partitioned into k disjoint clusters($|C| = k$) such that.

$$\frac{1}{k}\sum_{\forall x \in C} c(x) \quad (4)$$

is minimal? $c(x)$ is calculated like so:

$$c(x) = \frac{number\ of\ 'external' links}{number\ of\ 'internal' links + 1}$$

If so can such a partitioning take place in polynomial time?

We can take this one step further and define a whole family of problems, where the k clusters are not disjoint anymore and their intersections' cardinality can be up to a variable upper bound ranging from 1 to n. The question is still valid: Can there be an optimal partitioning in regards to cluster cohesion for k clusters with variable intersection cardinality (Figure 12)?

Existing research by Balalau et al.[16] has considered the related problem of finding at most k subgraphs with maximum total aggregate density(Given an undirected graph $G = (V, E)$, they define its density $p(G)$ to be $\frac{|E|}{|V|}$) with an upper bound on the pairwise Jaccard coefficient between the sets of nodes of the subgraphs. They conclude that this problem is NP-hard. This issue differs from ours as it tries to maximize the sum of densities of k subgraphs while we try to minimize the average cohesion of k clusters. We want the maximum amount of internal edges and the minimum amount of external edges simultaneously.

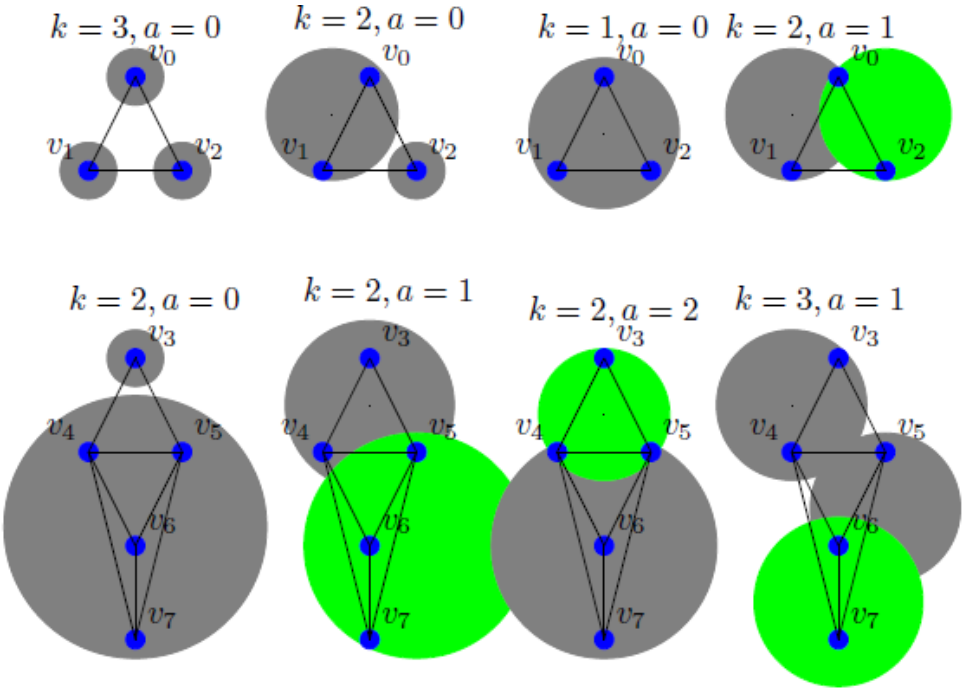

Figure 12: These are indicative partitionings of graphs that help make the problem clearer. This does not mean, that they are optimal in each case. Above each graph we state the number of clusters (k) as well as the maximum cardinality of cluster intersections (a).

6. CONCLUSION AND FUTURE WORK

This work was the result of a concerted effort to analyse and compare two methods used to solve the prefetching problem in graphs. The first defines cluster cohesion and looks for dense subgraphs and a second one that tries to keep the surfer satisfied by fetching all the possible nodes he is going to visit using the graph ‘s surveillance number. We also provide the pseudo-code for both methods and explain the similarities and differences they have both in theory and in practice.

Along the way, we provide a more complete definition for cluster cohesion that accounts for edge cases. Finally, we define an original problem, which consists of partitioning a graph into a predefined amount of disjoint clusters of optimal average cohesion.

Before we conclude we ask some questions for future work:

- Can the graph be partitioned into k disjoint clusters so that their average cohesion is optimal and if so can it be done in polynomial time?

- Can the graph be partitioned into k clusters with intersections of cardinality up to a factor a ranging from 1 to n so that their average cohesion is optimal and if so can it be done in polynomial time?

- If we know the surveillance number of a graph can we deduce anything about its optimal cohesion? Can the surveillance number be used as an upper/lower bound of the optimal cluster cohesion and vice versa?

- Can a combination of the surveillance game approach and C3i be used to create a better clustering overall in the case of trees?

ABBREVIATIONS – ACRONYMS

CDN	Content Distribution Network
ISP	Internet Service Provider
DoS	Denial of Service
C3i	Correlation Clustering Communities identification
DNS	Domain Name System

REFERENCES

1. A. Sidiropoulos, G. Pallis, D. Katsaros, K. Stamos, A. Vakali and Y. Manolopoulos, Prefetching in Content Distribution Networks via Web Communities Identification and Outsourcing, *World Wide Web Journal (Springer)*, vol. 11, no. 1, Mar. 2008, pp. 39-70.

2. F. V. Fomin, F. Giroire, A. Jean-Marie, D. Mazauric and N. Nisse, "To Satisfy Impatient Web surfers is Hard", *Proc. 6th Int'l Conf. on FUN with Algorithms (FUN 12)*, Springer, 2012, pp.166-176. <hal-00704201>

3. F. Giroire, I. Lamprou, D. Mazauric, Nicolas Nisse, S. Pérennes and R. Soares, Connected Surveillance Game, *Journal of Theoretical Computer Science (TCS)*, *Elsevier*, vol. 584, Jun. 2015, pp.131-143. <hal-01163170>

4. S.E. Schaeffer, Graph Clustering, *Computer Science Review,* vol. 1, no. 1, Aug. 2007, pp. 27-64.

5. J. Xu, J. Liu, B. Li and X. Jia, Caching and Prefetching for Web Content Distribution, *Computing in Science and Engineering (CiSE)*, vol. 6, no. 4, Jul. 2004, pp. 54-59.

6. N.C Zakas, "How content delivery networks (CDNs) work", Nov. 2011; https://www.nczonline.net/blog/2011/11/29/how-content-delivery-networks-cdns-work/ [Προσπελάστηκε 15/5/16]

7. B. Marrivada, "Consumer reaction to a poor online shopping portal", Study, Forrester Consulting, 2 Sep. 2010; http://www.slideshare.net/mbharath/consumer-reaction-to-a-poor-online-shopping-portal [Προσπελάστηκε 20/5/16]

8. "CDN Market Projected to US$ 12 Billion by 2019", Pune, India, Apr. 2014; http://www.satellitemarkets.com/market-trends/cdn-market-projected-us-12-billion-2019 [Προσπελάστηκε 20/5/16]

9. T. Watson, "Why the CDN Market is Expected to Triple by 2019", Nov. 2014; http://www.business2community.com/tech-gadgets/cdn-market-expected-triple-2019-01074693 [Προσπελάστηκε 20/5/16]

10. M. Claeys, D. Tuncer, J. Famaey, M. Charalambides, S. Latre, G. Pavlou and F. De Turck, Hybrid Multi-tenant Cache Management for Virtualized ISP Networks, *Journal of Network and Computer Applications (JNCA),* vol. 68, no. C, Jun. 2016, pp. 28-41.

11. P. Hurley, "The 4 Keys to Telco CDN Success", Whitepaper, Skytide, 16 Feb. 2011;http://www.slideshare.net/skytide/the-4-keys-to-telco-cdn-success [Προσπελάστηκε 15/5/16]

12. F. V. Fomin, F. Giroire, A. Jean-Marie, D. Mazauric and N. Nisse, To Satisfy Impatient Web surfers is Hard. [Research Report] RR-7740, LIRMM; INRIA. 2011, pp.20. <inria-00625703v2>

13. M.R. Garey, D.S. Johnson and L. Stockmeyer, Some simplified NP-complete graph problems, *Theoretical Computer Science,* vol. 1, no. 3, Feb. 1976, pp. 237 –267.

14. N. Laoutaris, V. Zissimopoulos and I. Stavrakakis, On the Optimization of Storage Capacity Allocation for Content Distribution, *Computer Networks : The International Journal of Computer and Telecommunications Networking*, vol. 47, no. 3, Feb. 2005, pp. 409-428.

15. N. Laoutaris, O. Telelis, V. Zissimopoulos and I. Stavrakakis, Distributed Selfish Replication, *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 12, Dec. 2006, pp. 1401-1413.

16. O. D. Balalau, F. Bonchi, T. H. Chan, F. Gullo and M. Sozio, "Finding subgraphs with maximum total density and limited overlap", *Proc. 8th ACM Int'l Conf. on Web Sear. and Data Min. (WSDM 15)*, 2015, pp. 379-388

17. P. Wendell and M. J. Freedman, "Going viral: Flash crowds in an openCDN", *Proc. 11th ACM SIGCOMM Int'l Meas. Conf. (IMC 11)*, New York, NY, USA, 2011, pp. 549-558.

18. I. Ari, B. Hong, E.L. Miller, S.A Brandt and D.D.E Long, "Managing Flash Crowds on the Internet", *Proc. 11th IEEE/ACM Int'l Symp. on Model., Anal. and Sim. of Comp. and Telecomm. (MASCOTS 03)*, 2003, pp. 246-249.