Faster heuristics for graph burning

Rahul Kumar Gautam[1] · Anjeneya Swami Kare[1] · Durga Bhavani S.[1]

Abstract

Graph burning is a process of information spreading through the network by an agent in discrete steps. The problem is to find an optimal sequence of nodes that have to be given information so that the network is covered in least number of steps. Graph burning problem is NP-Hard for which two approximation algorithms and a few heuristics have been proposed in the literature. In this work, we propose three heuristics, namely, Backbone Based Greedy Heuristic (BBGH), Improved Cutting Corners Heuristic (ICCH), and Component Based Recursive Heuristic (CBRH). These are mainly based on Eigenvector centrality measure. BBGH finds a backbone of the network and picks vertex to be burned greedily from the vertices of the backbone. ICCH is a shortest path based heuristic and picks vertex to burn greedily from best central nodes. The burning number problem on disconnected graphs is harder than on the connected graphs. For example, burning number problem is easy on a path where as it is NP-Hard on disjoint paths. In practice, large networks are generally disconnected and moreover even if the input graph is connected, during the burning process the graph among the unburned vertices may be disconnected. For disconnected graphs, ordering the components is crucial. Our CBRH works well on disconnected graphs as it prioritizes the components. All the heuristics have been implemented and tested on several bench-mark networks including large networks of size more than 50K nodes. The experimentation also includes comparison to the approximation algorithms. The advantages of our algorithms are that they are much simpler to implement and also several orders faster than the heuristics proposed in the literature.

1 Introduction

An application of information spread can be seen in the public health campaigns. For instance in the case of covid pandemic, the health workers are struggling to get the information reach every house in order to spread awareness and prevent the spread of coronavirus. Consider the scenario of one health-worker endeavour. She contacts a family in the village and explains the precautions to be taken up to protect oneself from the contagious infection and persuades that family to help spread awareness among their acquaintances. We assume that all the members who can be influenced by a family are covered in one step of the awareness campaign. In the burning number context, all those nodes that are covered are called as the 'burning' nodes. Next, she has to approach another new family outside this circle of influence to spread the message. The whole process happens in discrete steps. Hence it is important to detect an optimal sequence of families to be approached by the health worker that would ensure coverage of the entire village for propagating the information in minimum time. In paper [25], a similar application of passing information in emergency situations is discussed. Suppose a piece of information has to be sent to all the nodes in a network by a satellite. The satellite informs nodes (hubs) in discrete time steps. Once a node is informed it is deemed to be in informed state. All the nodes that receive the information communicate to each of their neighbours at the next time instant. Assume that the communication is

✉ Anjeneya Swami Kare
askcs@uohyd.ac.in

Rahul Kumar Gautam
19mcpc06@uohyd.ac.in

Durga Bhavani S.
sdbcs@uohyd.ernet.in

1   School of Computer and Information Sciences,
     University of Hyderabad, Hyderabad, India

carried out in a parallel mode. Process stops when the nodes in the entire network are informed.

Graphs are popular representations that model the social networks of the real world. Let $G(V, E)$ be a graph where V is set of nodes depicting people, E denotes relationship among the nodes. Initially all the nodes are in unburned (uninformed) state. Given time steps $t_0, t_1, t_2 \cdots t_{b-1}$, at t_0 one node is set to fire from outside. It starts burning and spreads the fire to its neighbours in a step wise fashion. During the process of burning, it is assumed that either the node is set on fire directly, called as source of fire, or node is burning by catching fire from a neighbour or it is not yet burnt. At i^{th} time step, a new unburned node is set on fire from outside and all those nodes which have caught fire at t_{i-1}, burn their neighbours. The process stops when the entire graph is burning, that is, or all the nodes have received information. Thus the task is to find the minimum sequence of nodes that have to be chosen as sources of fire, that are directly burned from outside. It is desirable to spread the information through the network, or burn all the nodes of the network quickly. So, the goal is to minimize the number of sources. The minimum number of steps needed to burn the entire graph or the length of the optimal burning sequence is called the *burning number* of the graph. The burning number of the graph G is denoted by $bn(G)$. Let us consider the graph in Fig. 1. The vertex sequences [7, 4, 2, 1], [4, 7, 1] and [3, 6, 8] are valid burning sequences, where as the sequence [7, 4, 1] is not a burning sequence. The burning number of the graph is 3, as the graph does not have burning sequence of length less than 3.

There are a few problems related to graph burning proposed in the literature. K-centre problem [11], is one in which the K centres are chosen simultaneously as sources of fire. As a result the nodes burn in parallel and hence very quickly. The K-centres problem is also NP-hard. The Firefighter problem [10] is a complementary version of the graph burning problem, in which a firefighter protects a node to reduce spread of fire in the graph. At each time step, the firefighter selects a node through which he can protect maximum number of nodes. On the other hand, in graph burning, a node is selected in such a way that it can burn a maximum number of nodes. Therefore the firefighter defends and in graph burning the source of fire burns. Active influence spreading in social networks [7, 8,
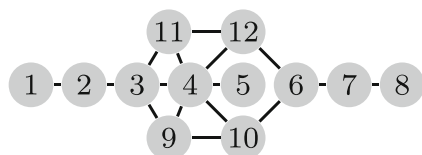
15] is the problem of selection of seed sets that can influence as many nodes as possible. In this paper, we propose three heuristics for the GRAPH BURNING problem. The proposed heuristics are tested on some real world data sets and their performance is compared to the existing heuristics.

2 Related work

Bonato et al. [3] introduced the GRAPH BURNING problem and the parameter *burning number*. They have studied the properties of GRAPH BURNING and proposed bounds for burning number. Bessy et al. [24] proved that the decision version of the GRAPH BURNING problem is NP-Complete. There has been lot of attention paid towards studying the GRAPH BURNING problem from theoretical point of view. The complexity and the algorithms for the GRAPH BURNING problem for special graph classes was studied in [4, 12, 21, 23, 24]. The characterization and bounds for burning number was studied in [1, 3, 4, 6, 13, 17, 19, 20]. Approximation algorithms for the burning number problem was studied in [5, 6, 13, 24]. Parameterized complexity of the GRAPH BURNING problem was studied in [14, 16]. Refer to the survey paper by Bonato [2] for more details of the state-of-the-art results on the burning number problem.

Šimon et al. [25] proposed heuristics for the GRAPH BURNING problem. They have studied the GRAPH BURN-ING problem empirically using both real world data sets as well as synthetic data sets. They proposed three heuristics based on Eigenvector centrality, namely Maximum Eigenvector Centrality Heuristic (MECH), Cutting Corners Heuristic (CCH), and Greedy Algorithm with Forward-Looking Search Strategy Heuristic (GFSSH). The MECH is a greedy heuristic, at each iteration it selects a (central) node with maximum eigenvector centrality. In CCH, at each iteration, first it finds a set of corner nodes of the graph, using these corner nodes, a set of central nodes are selected based on eigenvector centrality. Among these central nodes a best central node is selected using weighted aggregated sum product assessment (WASPAS) algorithm. In GFSSH a set of 20 central nodes are generated and at each iteration a best central node is selected by combining greedy heuristic and forward looking search.

Šimon et al. [25] also implemented the 3-approximation algorithm (3-APRX) of Bonato et al. [5] to compare the performance of their heuristics. They have tested the heuristics on some synthetic tree data sets also, however they did not implement the 2-approximation algorithm (2-APRX) for trees of Bonato et al. [5]. As stated in [5], the 2-approximation algorithm can also be used to compute an upper bound on burning number of any graph. If G is the original graph and T be any spanning tree of G then $bn(G) \leq bn(T)$. So the 2-approximation algorithm for trees

Fig. 1 An example graph, The vertex sequence [4, 7, 1] is an optimal burning sequence. The burning number of the graph is 3

can also be used to compute an upper bound on the burning number of the graph. However the computed upper bound need not be a 2-approximation of the original graph.

In their extended work, Šimon et al. [26] compared the effectiveness of greedy heuristic of [25] with 31 different centrality measures.

Recently, Farokh et al. [9] proposed six heuristics for the GRAPH BURNING problem. Their heuristics are not based on Eigenvector centrality. They call the vertices in the burning sequence as activators. The first four heuristics are based on different strategies to obtain the first activator and the rest of the activators. First activator is either a central node or it is selected randomly. The rest of the activators are selected such that each vertex has a unique activator. In other words reduce overlapping among the circle around the activators. The other two heuristics are based on diameter, DFS and BFS of the graph.

We propose three heuristics for the GRAPH BURNING problem. The heuristics are based on eigenvector centrality. We propose the following heuristics: Backbone Based Greedy Heuristic (BBGH), Improved Cutting Corners Heuristic (ICCH) and Component Based Recursive Heuristic (CBRH). We have also implemented both 3-approximation and 2-approximation algorithms of Bonato et al. [5]. We compare our implemented heuristics with GFSSH, the best performing heuristic of Šimon et al. [25]. We also compare performance of our algorithms with the results of Farokh et al. [9].

Note that both [25] and [9] tested their heuristics on smaller data sets. We test our heuristics on bigger data sets as well. For example, some of the data sets we used are DIMACS, BOSHLIB, Facebook blue verified pages friends network, DBLP-citation network and huge graphs with size more than 50,000 nodes like Gemsec-Deezer(HR) (music friendship network in Europe). All the three heuristics are performing equally well on all the data sets. Our heuristics are faster than the GFSSH of Šimon et al. [25] and efficient compared to heuristics of Farokh et al. [9]. Moreover our heuristics are easy to implement.

The rest of the paper is organized as follows: In Section 3, we discuss greedy heuristics. The Backbone Based Greedy Heuristic (BBGH) is discussed in Section 3.1. The Improved Cutting Corners Heuristic (ICCH) is discussed in Section 3.2. In Section 3.3, we discuss the Component Based Recursive Heuristic (CBRH). In Section 4, we discuss our results and some observations. We give conclusions in Section 5.

3 Proposed heuristics

We consider the following decision version of the GRAPH BURNING, which asks to check if the graph G can be burned in at most b time steps. If we have an algorithm for BURN-GRAPH(G, b), we can use binary search to compute minimum b for which the BURN-GRAPH(G, b) returns true. As the GRAPH BURNING is NP-hard, we can not expect to have an exact polynomial time algorithm for BURN-GRAPH(G, b). In this paper, we propose heuristics for BURN-GRAPH(G, b). The heuristics for BURN-GRAPH(G, b), if it returns $true$, it means that the algorithm is successful in finding a burning sequence of length at most b. If it returns $false$, it means that the algorithm failed to find a burning sequence of length at most b. Note that in the latter case, the graph can still be burned in at most b steps.

BURN-GRAPH(G, b):
Input: A graph $G = (V, E)$ and a positive integer b.
Question: Does the graph have a burning sequence of length at most b?

We propose three heuristics for BURN-GRAPH(G, b). First two are greedy in nature and the third one is a component based recursive algorithm. First we discuss the greedy algorithms and then the recursive algorithm.

At the outset, the process underlying the greedy algorithms is as shown in Algorithm 1. At each iteration, the function $getBestCentralNode()$ extracts an unburned vertex to burn next. The two heuristics differ in the procedure $getBestCentralNode()$.

Algorithm 1 Graph burning algorithm.

Input : $G = (V, E)$ and a positive integer b
Output: Returns true if algorithm finds a burning
           sequence of length at most b or false otherwise

1 BURN−GRAPH(G,b) **begin**
2    $G'(V', E') \longleftarrow G(V, E)$
3    $BS[0..b-1]$ ▷ An array to represent burning sequence
4   **for** $j \leftarrow 0\ to\ b-1$ **do**
5      $bestNode \longleftarrow getBestCentralNode(G', b-j-1)$
6      $BS[j] \longleftarrow bestNode$
7      $S \longleftarrow N_{G'}^{b-j-1}[bestNode]$
8      $V' \longleftarrow V' \backslash S$
9      $G' \longleftarrow G'[V']$
10   **if** $V' = \emptyset$ **then**
11   **for** $k \leftarrow 0\ to\ j$ **do**
12   print($BS[k]$)
13   **return** true
14   **return** false

3.1 Backbone Based Greedy Heuristic (BBGH)

We call a longest path starting at a node with minimum centrality value and containing nodes with high centrality values as the backbone path. A node on the backbone path can potentially burn more vertices. With this intuition we propose the Backbone Based Greedy Heuristic (BBGH). We extract the backbone path and for each vertex in the backbone path, we see how many vertices the vertex can burn, if we burn the vertex in the current time step. The vertex which leads to maximum number of burned vertices is chosen as the best central node. Here the crucial step is: how to extract the backbone path?.

Algorithm 2 Backbone based greedy heuristic.

Input : $G = (V, E)$ and a non-negative integer r
Output: Returns Best central node

1 getBestCentralNode(G, r) **begin**
2    $C[0..q-1] \longleftarrow getComponents(G)$
3    $bbPath[0..n-1]$ ▷ An array to represent vertices in backbone path
4    $bbLength \leftarrow 0$
5    $bbCenrality \leftarrow 0$
6   **foreach** $comp \in C$ **do**
7      $path[0..l-1] \leftarrow getBackbonePath(comp)$
8      $pathCentrality \leftarrow \sum_{v \in path}(centrality(v))$
9   **if** $bbLength = l$ and $pathCentrality > bbCenrality$ **then**
10        $bbCenrality \leftarrow pathCentrality$
11        $bbpath \leftarrow path$
12   **else if** $bbLength < l$ **then**
13        $bbLength \leftarrow l$
14        $bbCenrality \leftarrow pathCentrality$
15        $bbPath \leftarrow path$
16    $max \leftarrow 0$
17    $bestNode \leftarrow null$
18   **foreach** $vertex\ v \in bbpath$ in decreasing order of centrality **do**
19      $S \leftarrow N_G^r[bestNode]$
20   **if** $|S| > max$ **then**
21        $max \leftarrow |S|$
22        $bestNode \leftarrow v$
23   **return** bestNode

A backbone path is a longest path starting at a node with minimum centrality and containing nodes with high centrality values. To compute backbone path we use BFS traversal. We compute BFS tree rooted at a node with minimum centrality value. In this rooted tree we look at the nodes at highest depth, there can be more than one such node. For all these nodes we consider shortest path from the root to the node and compute average centrality of all the nodes in the path. A path with maximum average centrality is considered as the backbone path. The procedure $getBackbonePath()$ returns a path.

Let us take backbone path as an array $bbPath$. For each vertex $v \in bbPath$ in decreasing order of centrality values, compute $S = N_{G'}^r[v]$, set of all the vertices which are at a distance at most r from v. Then whichever node gives maximum $|S|$ value, will become the best central node. If the graph is disconnected, then backbone path of each component is extracted and which ever vertex of these backbone paths gives maximum $|S|$ value, we return that vertex as the best central node. The complete algorithm is shown in Algorithm 2. Working of the BBGH is shown in Table 1 with an example. Note that, for the graph given in Table 1, GFSSH of Šimon et al. [25] gives a burning sequence of size 5, where as our BBGH burns the graph in 4 time steps.

3.2 Improved Cutting Corners Heuristic (ICCH)

Šimon et al. [25], presented heuristic called Cutting Corner Heuristic (CCH). Their algorithm has $O(mn)$ time complexity. We present a similar heuristic which also runs in worst case $O(mn)$ time. However our algorithm is easy to implement and runs faster in practice as we avoid computation of average path length and call to weighted aggregated sum product assessment (WASPAS) method.

Let r be the number of time steps available to burn the graph. We start by computing the centrality values. Let u be the node with maximum centrality. We remove the r neighborhood of u, that is, all the vertices in the set $N_G^r[u]$ from the graph G. If the resulting graph is empty, then we return the vertex u as the best central node. Otherwise, let the resulting graph have q components, say $C[0], C[1], \cdots, C[q-1]$. For each component $C[i]$, $0 \le i < q$, we take the minimum centrality vertex (say v_i) of the component $C[i]$ and compute shortest path from u to v_i and let the shortest path be denoted by $P[i]$. We visualize $P[0], P[1], \cdots, P[q-1]$ as a matrix (multi-list), where each $P[i]$ is treated as a row in the matrix. Now for each column of the matrix, we pick r nodes in decreasing order of degree. If c is the number of columns of the matrix, we will get at most $r * c$ such nodes. For each of these nodes, we compute $S = N_G^r[.]$ value. Then whichever node gives maximum $|S|$ value, will become the best central node. The

process is depicted in the Algorithm 3. Working of the ICCH is shown in Table 2 with an example. From our results, we observe that our ICCH performs better than the CCH of Šimon et al. [25].

Algorithm 3 Improved cutting corners heuristic.

Input : $G = (V, E)$ and a non-negative integer r
Output: Returns Best central node

1 getBestCentralNode(G,r) **begin**
2      $v \longleftarrow \text{argmax}_{v \in V}\{centrality[v]\}$
3      $S \longleftarrow N_G^r[v]$
4      $C[0, 1, \cdots q-1] \longleftarrow getComponents(G')$
5   **if** $q = 0$ **then**
6   **return** v
7      $matrix[][]$ ▷ A two dimentional array (multi-list) to store vertices in the paths
8   **for** $j \leftarrow 0 \ to \ q - 1$ **do**
9         $src \longleftarrow \text{argmin}_{u \in C[j]}\{centrality[u]\}$
10        $path \longleftarrow getShortestPath(G, v, src)$
11        $matrix[j] \longleftarrow path$
12     $capacity \longleftarrow |S|$
13     $bestNode \longleftarrow v$
14   **for** *each column* $c \in matrix$ **do**
15        $topNodes \longleftarrow getToprNodesByDegree(c)$ ▷ This function returns a list of top r nodes in decreasing order of degree
16   **for** $w \in topNodes$ **do**
17           $S \longleftarrow N_G^r[w]$
18   **if** $capacity \leq |S|$ **then**
19              $capacity \longleftarrow |S|$
20              $bestNode \longleftarrow w$
21   **return** $bestNode$

3.3 Component Based Recursive Heuristic (CBRH)

If the graph is disconnected, choosing of a component to burn a vertex can make a difference. Let us consider the graphs in Figs. 2 and 3. These graphs have two components. Let us see the following criterion to select the component to burn a vertex.

1. **Component with maximum size:** If we choose component with maximum number of vertices, we need a burning sequence of length 4 to burn the graph in Fig. 2 and we need a burning sequence of length 4 to burn the graph in Fig. 3. However, the actual burning number of the graph in Fig. 3 is 3.

2. **Component with maximum path length:** If we choose component with maximum path length, we need a burning sequence of length 5 to burn the graph in Fig. 2 and we need a burning sequence of length 3 to burn the graph in Fig. 3. However, the actual burning number of the graph in Fig. 2 is 4.

For the graph given in Fig. 2, the Backbone Based Greedy Heuristic will choose the component with vertex set $\{1, 2, 3 \cdots, 9\}$ and vertex 5 is chosen as the best central node. Now we can see that we require 5 time steps to burn the graph. However, the burning number of the graph is 4. The optimum burning sequence chooses a vertex 20 from the second component. Note that even if graph is connected in the beginning, in the subsequent iterations the graph can be disconnected and this situation can arise. Therefore choosing a component is very crucial. Therefore the question is what is the criteria to select the best component of the graph. Ideally we should choose a component that has maximum burning number.

In the Component Based Recursive algorithm, we recursively run the Backbone Based Greedy Heuristic and which ever component leads to maximum burning number, we choose such a component. For each component the algorithm recursively estimates the burning number of the component. During the recursive calls, burning number computed for a component is stored in a dictionary to avoid redundant recursive calls. The component with maximum estimated burning number is selected and from which best central node is selected. The process is described in the Algorithm 4. For the example considered in Table 1, the trace of the algorithm is very similar that of Algorithm 2.

Our BBGH and ICCH fail to compute optimal burning sequence of either graph in Figs. 2 or 3. However, our CBRH computes optimal burning sequence for both the graphs. For the connected graph given in Fig. 4, our CBRH computes the optimal burning number but our other two heuristics, BBGH, ICCH, and GFSSH of Šimon et al. [25] fail to compute optimal burning sequence. This concludes the significance of our ICCH.

4 Results and discussion

We tested our heuristics on the following data sets:

1. The Network Data Repository [22]

   – Netscience
   – Polblogs
   – Reed98
   – Mahindas
   – Cite-DBLP

Algorithm 4 Component based recursive heuristic.

    Input : $G = (V, E)$, a positive integer b and a dictionary D

    Output: Returns the length of minimal burning sequence of size at most b or -1 if algorithm fails to find burning

   sequence of length at most b. Note that the algorithm prints the burning sequence as well.

1 `EstimateBurningNumber`(G, b, D) **begin**

2    $bn \longleftarrow -1$

3   **for** $i \leftarrow b$ *down to* 1 **do**

4        $G'(V', E') \longleftarrow G(V, E)$

5        $BS[0..b - 1]$ ▷ `An array to represent burning sequence`

6   **for** $j \leftarrow 0$ *to* $i - 1$ **do**

7            $C[0..q - 1] \longleftarrow getComponents(G')$

8            $bestComp \longleftarrow C[0]$

9            $max \longleftarrow -1$

10          $bg \longleftarrow -1$

11   **if** $q > 1$ **then**

12   **for** $k \leftarrow 0$ *to* $q - 1$ **do**

13   **if** $C[k] \in D$ **then**

14                     $bg \longleftarrow D[C[k]]$ ▷ D `is a dictionary with component as the key and`

                                                `burning number as the value`

15   **else**

16                     $bg \longleftarrow$ EstimateBurningNumber$(C[k], i, D)$

17                     $D[C[k]] \longleftarrow bg$

18   **if** $max < bg$ **then**

19                     $max \longleftarrow bg$

20                     $bestComp \longleftarrow C[k]$

21          $bbPath \longleftarrow getBackbonePath(\text{bestComp})$

22          $max \longleftarrow 0$

23          $bestNode \longleftarrow null$

24   **for** *each vertex* $v \in bbpath$ **do**

25            $S \longleftarrow N_{bestComp}^{i-j-1}[v]$

26   **if** $|S| > max$ **then**

27                $max \longleftarrow |S|$

28                $bestNode \longleftarrow v$

29          $BS[j] \longleftarrow bestNode$

30          $S \longleftarrow N_{bestComp}^{i-j-1}[bestNode]$

31          $V' \longleftarrow V' \backslash S$

32          $G' \longleftarrow G'[V']$

33   **if** $V' = \phi$ **then**

34            $bn \longleftarrow i$

35   break

36   **if** $V' \neq \phi$ **then**

37   **return** bn

38   **return** bn

Table 1 Trace of BBGH, the Algorithms 1 and 2

Original Graph (*G*)	
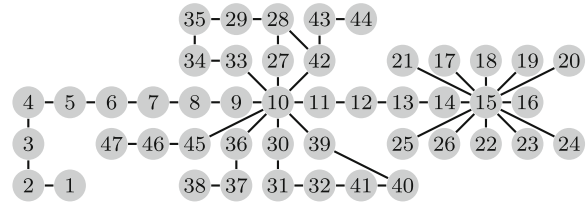
1st Iteration: Graph is connected and hence it has a single component. Backbone path *bbpath* = [16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1] and best central node is 10 for radius 3. Here, by radius we mean the number of remaining steps in the burning sequence.

After 1st Iteration	
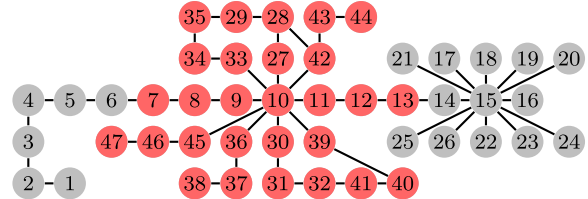
2nd Iteration: Graph has two components and backbone path *bbpath* = [6, 5, 4, 3, 2, 1] and best central node is 3 for radius 2.

After 2nd Iteration	
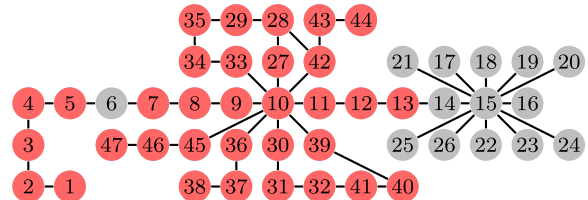
3rd Iteration: Graph has two components and backbone path *bbpath* = [16, 15, 14] and best central node is 15 for radius 1.

After 3rd Iteration	
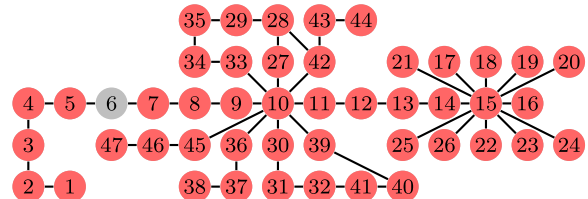
4th Iteration: Graph has one component and backbone path *bbpath* = [6] and best central node is 6 for radius 0.

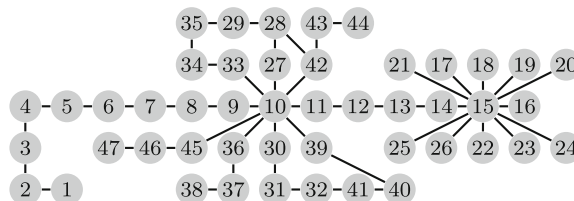| After 4th Iteration	
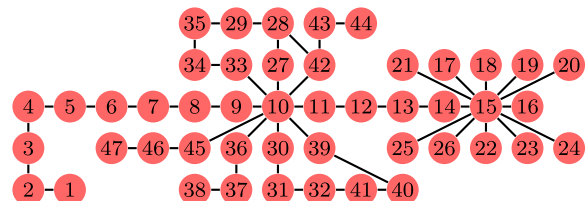
For each iteration vertices shown in red color are not part of the graph *G*. The estimated burning number of the graph is 4

2. Stanford large network dataset collection (SNAP Datasets) [18]

   – Chameleon
   – TVshow
   – Ego-Facebook
   – Squirrel
   – Politician
   – Government
   – Crocodile
   – Gemsec-Deezer(HR)

We also generated 100 random trees and tested the performance of the algorithms. All the heuristics are implemented in Python programming language. The algorithms have been implemented on a system with processor Intel Core $i5$, processor speed of 2.7 GHz having dual core and 8GB RAM.

Table 2 Trace of ICCH, the Algorithms 1 and 3

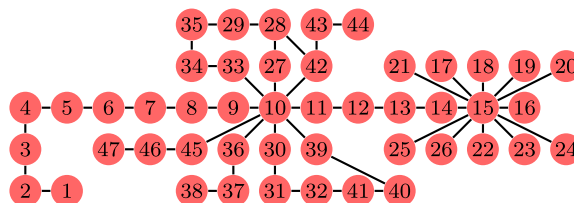Original Graph (*G*)

1st Iteration: graph is connected and hence it has a single component. We get node 10 as the next node to burn for radius 4.

After 1st Iteration
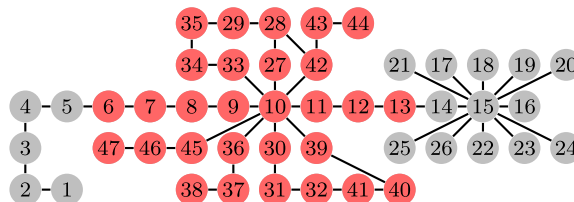
2nd Iteration: We get node 15 as the next node to burn for radius 3.

After 2nd Iteration
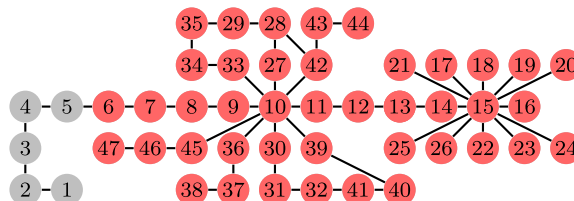
3rd Iteration: We get node 3 as the next node to burn for radius 2.

After 3rd Iteration

For each iteration vertices shown in red color are not part of the graph *G*. The estimated burning number of the graph is 5
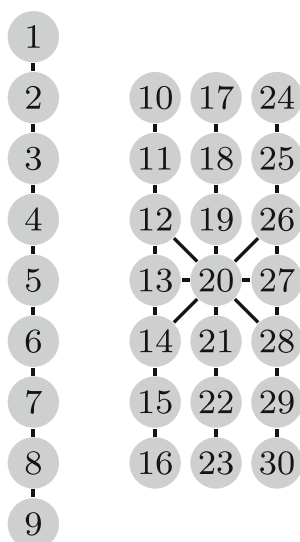
Fig. 2 An example disconnected graph. The burning number of the graph is 4

Fig. 3 An example disconnected graph. The burning number of the graph is 3
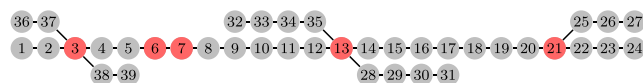
Fig. 4 An example graph, The vertex sequence [13, 21, 3, 7, 6] is an optimal burning sequence. The burning number of the graph is 5

We compare performance of our heuristics with those of [25] and [9]. The Table 3 shows the estimated values of the burning number for various algorithms. We have compared our results with 3-approximation and 2-approximation algorithms of Bonato et al. [5], GFSSH, the best performing heuristic of [25].[1] We have also listed the number of recursive calls made by our Component Based Recursive Heuristic. It can be observed that, at the outset the algorithm looks like an exponential algorithm, but in practice the number of recursive calls made is very less even for bigger graphs. Note that for all the social networking data sets and other data sets that have been considered in this paper, the diameter (radius) of the graph is very small. The burning number of a graph with radius r is at most $r + 1$. As the radius of the graphs is very small, improving the burning by even a small number is tough. From Table 3 we observe that our heuristics are competitive to the best heuristic of Šimon et al. [25]. Table 4 shows the running time comparison of our heuristics with that of GFSSH, the best performing heuristic of [25]. Our heuristics are faster than that of [25]. Note that both Šimon et al. [25] and Farokh et al. [9] tested their heuristics on smaller data sets. As our heuristics are faster, they take lesser time on even bigger data sets.

While comparing with heuristics of Farokh et al. [9], it can be seen that our heuristics give better results for some of the graphs. The Table 5 shows the comparison of performance of our heuristics with that of Farokh et al. [9].

In their extended work Šimon et al. [26] studied the effectiveness of 31 different centrality measures on the greedy heuristic of [25]. They have run the experimentation on only 3 data sets out of which 2 data sets are synthetic and the only standard data set they have used is the Netscience data set. Note that their experimentation is more on comparing different centrality measures and no new heuristics are proposed.

5 Conclusion

In this paper, we proposed three heuristics for GRAPH BURNING problem, namely, Backbone Based Greedy Heuristic (BBGH), Improved Cutting Corners Heuristic (ICCH), and Component Based Recursive Heuristic (CBRH). Firstly, we show a need for each heuristic by constructing the required example graphs. That is, we show a graph for which BBGH finds a better burning number compared to GFSSH, a graph on which ICCH finds a better burning number than BBGH; and finally need for the recursive algorithm of CBRH, where CBRH manages to find a better burning number than the other heuristics.

[1] The code is obtained from the authors and executed on our machine.

Table 3 Comparison of estimated burning number of approximation algorithms [5], CCH and GFSSH of [25] and our heuristics

Network source	Name	|V|	|E|	3-APRX	2-APRX	GFSSH [25]	CCH [25]	BBGH	ICCH	CBRH	CBRH Calls
Network data repository	Netscience	379	914	12	10	7*	7*	7	7	7	23
	Polblogs	643	2K	9	10	6	6	6	6	6	8
	Reed98	962	18K	6	8	4	4	4	4	4	46
	Mahindas	1258	7513	9	8	5*	5*	5	5	5	68
	Cite-DBLP	12.6K	49.7K	120	82	41	41	41	41	41	146
SNAP Data set	Chameleon	2.2K	31.4K	9	10	6	6	6	6	6	43
	TVshow	3.8K	17.2K	18	16	10	10	10	10	10	49
	Ego-Facebook	4K	88K	9	6	4	4	4	4	4	110
	Squirrel	5K	198K	9	10	6	6	6	6	6	19
	Politician	5.9K	41.7K	12	12	7	7	7	7	7	9
	Government	7K	89.4K	9	10	6	6	6	6	6	7
	Crocodile	11K	170K	12	10	6	6	6	6	6	40
	Gemsec-Deezer(HR)	54K	498K	12	12	7	7	7	7	7	92
Randomly generated	Barabasi-Albert (10 graphs)	1K	3K	6	8	4.9	4.9	4.9	4.9	4.9	8
	Erdos-Renyi (10 graphs)	1K	6K	6	8	5	5	5	5	5	1

The last column shows the number of recursive calls made by CBRH. *-For Netscience and Mahindas data sets, [25] quoted burning number as 6, however when we run their program on our machine we got 7 and 5 respectively

Table 4 Comparison of running times of our heuristics with CCH and GFSSH of [25]

Network Source	Name	|V|	|E|	GFSSH [25]	CCH [25]	BBGH	ICCH	CBRH
Network data repository	Netscience	379	914	2m	3s	<1s	<1s	1s
	Polblogs	643	2K	3s	3s	<1s	1s	2s
	Reed98	962	18K	5s	6s	3s	3s	5s
	Mahindas	1258	7513	6s	7s	<1s	3s	23s
	Cite-DBLP	12.6K	49.7K	3m 8s	3m 20s	39s	22s	2m
SNAP Data set	Chameleon	2.2K	31.4K	25s	27s	20s	16s	16s
	TVshow	3.8K	17.2K	30s	26s	7s	22s	15s
	Ego-Facebook	4K	88K	1m	1m	17s	16s	22s
	Squirrel	5K	198K	3m 5s	2m	40s	34s	1m 40s
	Politician	5.9K	41.7K	1m	52s	14s	32s	17s
	Government	7K	89.4K	1m 13s	1m 17s	20s	50s	32s
	Crocodile	11K	170K	5m	3m 17s	2m 36s	42s	4m
	Gemsec-Deezer(HR)	54K	498K	1h 20m	49m	2m 36s	7m	47m
Randomly generated	Barabasi-Albert(10 graphs)	1K	3K	1m 10s	1m	10s	10s	20s
	Erdos-Renyi(10 graphs)	1K	6K	1m 40s	1m 30s	10s	5s	30s

Table 5 Comparison of estimated burning number of heuristics of [9] and our heuristics

Name	|V|	|E|	Max.deg	ctr-Half	ctr-far	Rand-Half	Rnd-Far	DFS-path	D-BFS-Path	GFSSH [25]	CCH [25]	BBGH	ICCH	CBRH
c-fat200-1	200	1.5k	17	11	8	9	**7**	8	8	7	7	7	7	7
c-fat200-2	200	3.2k	34	6	6	6	5	**5**	6	5	5	5	5	5
c-fat200-5	200	8.4k	86	4	4	4	4	4	**3**	3	3	3	3	3
c-fat500-1	500	4.4k	20	12	11	12	**10**	15	17	9	9	9	10	9
c-fat500-10	500	46k	188	4	4	4	**4**	4	4	3	3	3	3	3
c-fat500-2	500	9.1k	38	9	8	9	8	11	**8**	7	7	7	7	7
c-fat500-5	500	23k	95	6	6	6	6	**5**	6	5	5	5	5	5

We show through extensive experimentation that BBGH turns out to be the fastest among all the heuristics and several orders faster than the heuristic GFSSH of Šimon et al., which is one of the latest heuristics proposed for this problem as shown in Table 4. To give an example, on the largest network of the benchmark data set with size 54K, BBGH gave burning number of 7 in $2m$ $36s$ as compared to $1h$ $20m$ taken by GFSSH. ICCH follows as a close second by delivering the result in 7 minutes. Lastly, the recursive heuristic though is slower than our other heuristics it runs faster than GFSSH on most of the large data sets. It shows a way to prioritize the ordering of selection of components in order to obtain optimal burning number. Further, we also show the superior results obtained by our proposed heuristics on other data sets experimented by Farokh et al. in Table 5.

We feel that the techniques used in the paper can be extended to active influence spreading problems like, Target Set Selection, Perfect Seed Set, and Perfect Evangelic Set problems.

References

1. Bessy S, Bonato A, Janssen J, Rautenbach D, Roshanbin E (2018) Bounds on the burning number. Discret Appl Math 235:16–22
2. Bonato A (2020) A survey of graph burning
3. Bonato A, Janssen J, Roshanbin E (2014) Burning a graph as a model of social contagion. In: Algorithms and models for the web graph, WAW 2014, vol 8882. Springer, pp 13–22. Lecture Notes in Computer Science
4. Bonato A, Janssen J, Roshanbin E (2016) How to burn a graph. Internet Math 12(1-2):85–100
5. Bonato A, Kamali S (2019) Approximation algorithms for graph burning. In: Theory and applications of models of computation. TAMC 2019. Springer, pp 74–92. Lecture Notes in Computer Science
6. Bonato A, Lidbetter T (2019) Bounds on the burning numbers of spiders and path-forests. Theor Comput Sci 794:12–19
7. Cordasco G, Gargano L, Rescigno A, Vaccaro U (2016) Evangelism in social networks: Algorithms and complexity. Networks 71(4):346–357
8. Cordasco G, Gargano L, Rescigno AA (2019) Active influence spreading in social networks. Theor Comput Sci 764:15–29
9. Farokh ZR, Tahmasbi M, Tehrani ZHRA, Buali Y (2020) New heuristics for burning graphs. arXiv:2003.09314
10. Finbow S, Macgillivray G (2009) The firefighter problem: A survey of results, directions and questions. Australas J Combin [electronic only] 43:57–77
11. Garcia J, Menchaca R, Sanchez J, Menchaca R (2018) Local search algorithms for the vertex k-center problem. IEEE Lat Am Trans 16(6):1765–1771
12. Gupta AT, Lokhande S, Mondal K (2020) NP-completeness results for graph burning on geometric graphs. arXiv:2003.07746
13. Kamali S, Miller A, Zhang K (2020) Burning two worlds: Algorithms for burning dense and tree-like graphs. In: Theory and practice of computer science, SOFSEM 2020. Springer, pp 113–124. Lecture Notes in Computer Science
14. Kare AS, Reddy IV (2019) Parameterized algorithms for graph burning problem. In: International workshop on combinatorial algorithms. Springer, pp 304–314
15. Kempe D, Kleinberg J, Tardos E (2003) Maximizing the spread of influence through a social network. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining (KDD'03). pp 137–146
16. Kobayashi Y, Otachi Y (2020) Parameterized complexity of graph burning. arXiv:2007.08811
17. Land MR, Lu L (2016) An upper bound on the burning number of graphs. In: International workshop on algorithms and models for the web-graph. Springer, pp. 1–8
18. Leskovec J, Krevl A (2014) SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data
19. Liu H, Zhang R, Hu X (2019) Burning number of theta graphs. Appl Math Comput 361:246–257
20. Mitsche D, Prałat P, Roshanbin E (2018) Burning number of graph products. Theor Comput Sci 746:124–135
21. Mitsche D, Prałat P, Roshanbin E (2017) Burning graphs: A probabilistic perspective. Graphs Combin 33(2):449–471
22. Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence. AAAI Press, p. 4292–4293
23. Sim K, Tan TS, Wong K (2017) On the burning number of generalized petersen graphs. Bull Malaysian Math Sci Soc 41:1657–1670
24. Stéphane Bessy S, Bonato A, Janssen J, Rautenbach DR, Roshanbin E (2017) Burning a graph is hard. Discret Appl Math 232:73–87
25. Šimon M., Huraj L, Luptáková I, Pospíchal J (2019) Heuristics for spreading alarm throughout a network. Appl Sci 9: 3269
26. Šimon M., Huraj L, Luptáková I, Pospíchal J (2019) How to burn a network or spread alarm. MENDEL 25(2):11–18