Critical Node Set Using Ant Colony Optimization

Ashwin Padmanabhan
Department of Industrial Engineering
Purdue University
West Lafayette, IN, USA
padmana6@purdue.edu

***Abstract*—This work deals with the critical node detection (CNP) by use of Ant-Colony based optimization techniques. Ant-Colony optimization is naturally well suited to such problems and is computationally inexpensive. CNP is a very recently proposed problem, which involves detecting and deleting k nodes in a Graph G such that the fragmentation in the resulting sub-graph is maximized. In order to test the proposed heuristics 4 benchmark problems from a previous work are used which utilize Erdos–Renyi, Watts–Strogatz, Forest Fire and Barabasi–Albert models. Each of these models create graphs with different structural properties which give variation in problem difficulty. I find that the suggested algorithm produces comparable results to previous works.**

***Index Terms*—Ant Colony Optimization, Critical Node Set, Heuristic optimization, Graph Fragmentation,**

I. Introduction

Given an undirected, unweighted input graph $G(V, E)$, and an integer k, the goal of the Critical Node Set Problem (CNP), is to find the set $S \subseteq V$ such that, the fragmentation of the resulting graph, $G' = G(V/S, E/\{(i,j) \ : \{i,j\} \not\subset V/S\}$ is maximized. This problem is fairly new and the first related work in this subject was done in [11] which focused more on edge deletion rather than node deletion [1]. Broadly, detection of critical nodes in a graph shed light on the structural properties of the network [8].

CNP has many applications, especially in robustness of communication networks, network immunization, isolation of terrorist networks, among others [1] [8] [4] [5]. CNP is known to be NP-Complete [2]. Work on CNP has picked up momentum only recently [8] and a few strategies have been suggested to solve this problem, such as Local Search assisted GA [1], heuristics based algorithms [4] [2] among others.

I view the CNP problem as a network of roads on which there are certain junctions (nodes) and edges can be thought of as the roads originating from these junctions. The critical nodes in this network are the crucial roads such as bridges in the road network. This analogy came to me while watching a WWII documentary where Allied forces specifically targeted crucial bridges to disrupt information and resource flow of the Axis powers. This analogy, in my view, is further reinforced by the fact that by eliminating these bridges the Axis powers were often forced to take a longer route around a river or mountain that the bridge was build on. Meaning, the two points are not truly disconnected, but just so far away that they are virtually so [4].

Thanks to Dr. Mario Ventresca for his guidance and support

In this work, I devise mainly two methods based on Ant colony optimization to generate approximate solutions to CNP. The first approach relies on use of Ant Colony Optimization, wherein, ants deposit pheromone on nodes that they visit whereby over several iterations, more pheromone accumulates on the critical nodes which are then become candidate nodes to be removed. The second approach augments the Ant-Colony optimization algorithm whereby the graph is simulated as a network of Pipes with water flowing in them, with nodes acting as valves. Nodes with higher flow rate start off with a higher pheromone (or are directed by a Daemon) value and an Ant colony optimization is run on them. The motivation for the second approach being that for dense networks, an ant will have a higher probability of getting out of the these dense areas and pass through the critical nodes.

In order to test the results of the proposed algorithm, 4 benchmark problems from [8] are used. These 4 problems are created using Barabasi–Albert [3], Erdos–Renyi [6], Watts–Strogatz [9] and Forest Fire [7] random network generation algorithms. [8].

Organisation of the rest of the report is as folows: Section 2, "Methodology" gives the algorithm for the methods. Section 3, "Results and Evaluation" talks about the results for various randomly generated graphs as well as the 4 problem instances mentioned above. Section 4 gives Conclusions and possible future works.

II. Methodology

CNP is relatively, a recent problem, and a lot of work has not been done on this. Hence, there isn't a definitive guide to the structure an algorithm must follow. This work, as stated above, proposes two Ant-Colony based methods to solve the Critical Node Set problem.

A. Method 1

The first method (Method 1) uses is an unbiased-initialization Ant-Colony optimisation technique to solve CNP. Given an input, undirected and unweighted graph, G, this method involves the use of ants starting at different nodes in the graph to make decisions at each node as to which node

they must jump to next based on the pheromones dropped on these nodes by previous ants. It is hypothesised that the nodes in the graph that are critical will see higher ant traffic than other nodes. The k nodes with the highest pheromone in the end are selected to be the critical nodes.

In this method, we are given an input graph G and an integer k. Ants equal to the number of nodes in G are created and then randomly distributed to different nodes in the graph. All nodes are initialized with a fixed amount of Pheromone. In each iteration of the algorithm, ants in each node, move to an adjoining node, based on the a transition rule which is a function of pheromone values of the nodes. Transition probability for an ant k presently in node i moving to an adjoining node j is given by in equation 1. Below equation is inferred from [12]:

$$p_{ij}^{k} = \frac{\tau_j^{\alpha} * \eta_{ij}^{\beta}}{\sum_{j \in allowed_k} \tau_k^{\alpha} * \eta_{ik}^{\beta}} \tag{1}$$

In equation 1, τ_j is the pheromone value of node j at the given iteration, η_{ij} is an artificial kicker to direct the ants in the right direction, based on prior information. In this method, this is assumed to be 1. α and β can be thought of as weights assigned to Pheromone based/Daemon based decision process, respectively. Based on this transition rule, ants progress to a new node and deposit a fixed amount of pheromone, Γ, on these nodes. Once all ($|V|$) ants make one step each, some pheromone is evaporated from each node. The pheromone update rule is given in equation 2. This is a slightly modified version of the pheromone update rule for solving TSP given in [12].

$$\tau_i(t+1) = \textbf{Max}(\rho * \tau_i(t) + \Gamma, \gamma) \tag{2}$$

In equation 2, $0 \leq \rho \leq 1$, which is the fraction of pheromone retained in each node after an iteration and γ is the minimum level of pheromone in each node. This ensures that all nodes have a non-zero probability of being visited. Pseudo-code given below:

Algorithm 1 Ant Colony Optimization for CNP (Method 1)

```
procedure ACO(G, k)
    Number of ants ← Number of nodes in Graph
    Ants in each node ← Distribute |V| ants to nodes
    Initialize pheromone to each node
    for iternum < IternationMax do
        for Node in Graph do
            for Ant in Node do
                Ant jumps to new node
                Deposit pheromone on new node
                Update number of ants in source & target node
        Evaporate Pheromone
        Update Transition Probability Matrix
    return k nodes with highest pheromone
```

1) System Parameter Setting for Method 1: : The following are the parameters used in all the analysis in this work, unless stated otherwise:
The number of ants in the system are initialized to the number of nodes in the graph G. A uniform distribution is used to distribute the ants to various nodes. $\alpha = 2$, $\eta = 1$, Initial Pheromone levels $\tau_i(t=0) = 10 \quad \forall \quad i$, Evaporation (retension) rate, $\rho = 0.9$, Minimum Pheromone level , $\gamma = 5$, Pheromone dropped at each visit, $\Gamma = 2$. This algorithm is run usually for 10 iterations.

B. Method 2

Method 1 performs poorly for highly dense graphs with high clustering coefficient, such as the cases shown in Figure 4a and 4c. A deep dive analysis shows that often due to highly dense networks, ants are unable to escape these regions and pass through critical nodes. The motivation for Method 2 is to guide the ants in the right direction, either by use of a Daemon, or by initially setting weighted pheromone values to nodes in the graph. Method 2 is a strategy to sensibly choose these weights. Method 2 is inspired from a network water pipeline. The edges are pipes and the nodes are valves that are responsible for the equal distribution of flow to out-flowing pipes. All crucial nodes are hypothesised to have high flow rate. That is, given a constant flow of water coming into the network at a given node (the entry node), the flow is distributed at each node and it is theorized that the nodes with the highest flow-rate are candidates to be a crucial node. The flow at a given node is averaged over cases sequentially considering all nodes as entry nodes. To simulate this experiment I use a breadth-first search of nodes and and sequentially distribute flow into out-flowing edges. The flow-rates for each node are then used either weight η in equation 1 or use a weighted initialization of pheromone, which is then fed into Method 1.

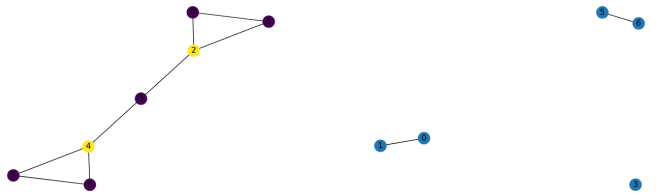
Pseudo-code given below:

Algorithm 2 Method 2: Breadth-First based node ranking

```
procedure flowrate(G(V,E))
    resdict ← Initialize dictionary of structure {Node#:0}
    for entrynode in {V} do
        flowratedict ← Initialize dict {Node#:0}
        traverse ← Empty list
        Append entrynode to traverse
        Append 'inflowrate' to flowratedict[entrynode]
        Append neighbours of entrynode to traverse
        Append value of flowrate of neighbours of entrynode
        while len(Traverse) ≤ |V| do
            for node in Traverse do
                for neighbours of nodes do
                    Append neighbours to traverse
                    Update flowratedict with flowrates of neighbours
                Update resdict
    return Average(resdict)
```

III. Results and Evaluation

To start off, Method 1 was tested on simple Barbell graphs, with trivial critical nodes, identifiable by visual inspection. The network is shown in Figure 1 below. Nodes deleted are represented in yellow.

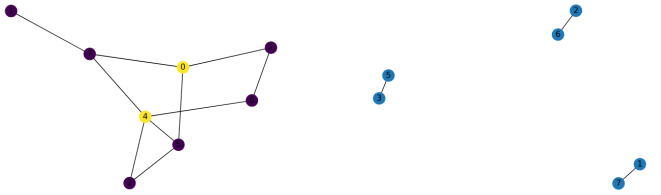


(a) 3-1 Barbell input graph (b) 3-1 Barbell output graph

Fig. 1: ACO implementation for $k = 2$ on a 3-1 Barbell Graph

Now, the algorithm was further tested for several random Sedge-wick graphs. One outcome shown in Figure 2 with $k = 2$.

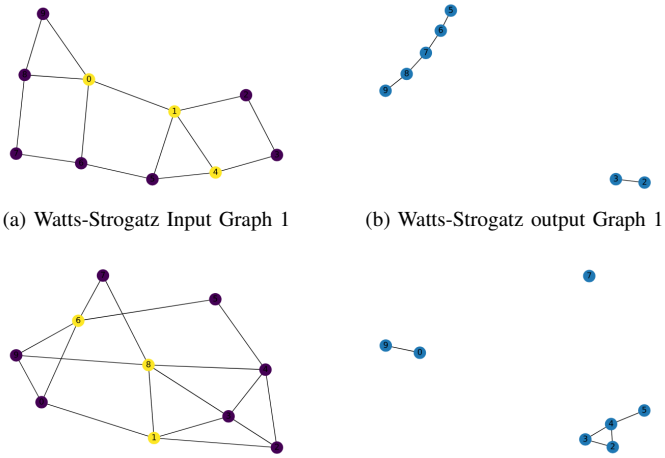
Method 1 was next implemented on several randomly graphs



(a) Input Sedgewick Graph (b) Output Sedgewick Graph

Fig. 2: ACO implementation for $k = 2$ on a Sedge-Wick Graph

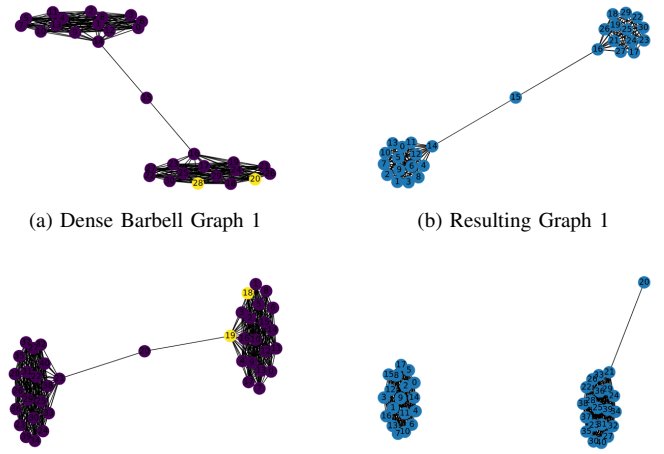
generated using the Watts–Strogatz model, leveraging it's properties of creating highly clustered networks [10]. As an example, two random graphs with 10 nodes were generated and the Ant Colony was run with input $k = 3$. The network is shown in Figure 3 below. Nodes deleted are represented in yellow. The results seem reasonable and the I note that the algorithm is robust in identifying critical nodes for such graph networks. Furthermore, it was suspected that this methodology would not work for highly dense graphs such as the one shown in Figure 4a and Figure 4c. The reasoning being that ants would get stuck in these dense areas and not reliably pass through the critical nodes. To test this, I use a dense Barbell Graph, with trivial critical nodes. I find that though the algorithm does a pretty decent job at find these critical nodes many times, there are instances when it does not converge to the optimal, as shown in Figure 4 below. We see in Figure 5a that Method 2 gives reasonable results for dense barbell graphs. In Figure 5b we see that though only two clusters are created, it results in sub-graphs that have high path lengths. It is noteworthy that two networks do not need to actually belong to different clusters to be disconnected. Distances between nodes are so high that the nodes are virtually disconnected [4].



(a) Watts-Strogatz Input Graph 1 (b) Watts-Strogatz output Graph 1

(c) Watts-Strogatz Input Graph 2 (d) Watts-Strogatz output Graph 2

Fig. 3: ACO implementation for $k = 3$ on Watts-Strogatz Graph with 10 nodes



(a) Dense Barbell Graph 1 (b) Resulting Graph 1

(c) Dense Barbell Graph 2 (d) Resulting Graph 2

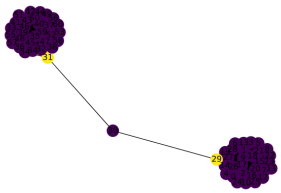
Fig. 4: ACO implementation for $k = 2$ on Dense Barbell Graph.

A. Benchmark Problems:

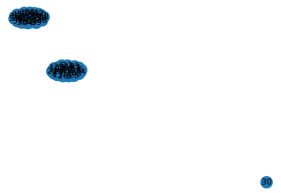
To gauge the performance of the algorithms, a subset of problem instances from [8] are chosen as baselines. The instances chosen are WS250, ER250, FF250 and BA1000. The reader can find more information on the problem instances in [8].

B. Running time:

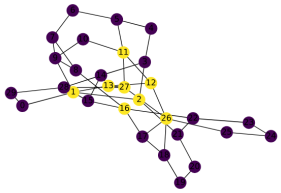
In this section, I look at the average run time of Method 1 and Method 2 on the aforementioned benchmark problem instances. Time to load the instances into NetworkX objects is not accounted for, and only the run time of the algorithms is considered. The algorithm is run 10 times, and the average of the run time is reported. All runs are done with $k = 70$ and



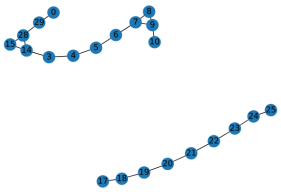
(a) Dense 30-1 Barbell Graph



(b) Result for 30-1 Barbell Graph using Method 2



(c) Dense Watts-Strogatz Graph



(d) Result for Watts-Strogatz Graph using Method 2

Fig. 5: Method 2 implementation with $k = 2$ and $k = 8$ for Barbell and Watts-Strogatz graph, respectively.

Number of Ants equal to the number of nodes in the input graph. Since ER250 is a disconnected graph, Method 2 can't be applied since it used flow rate computations, which can be done only on a non-fragmented input graph. All experiments were conducted on an Intel i5 (Dual-core, single thread 2.3 GHz) machine with 8 GB RAM, with the algorithms being programmed in Python 3. Running time results are tabulated in Table I.

Run Time		
Instance Name	Method 1 (s)	Method 2 (s)
WS250	20.4	20.5
ER250	18.77	NA
FF250	20.1	20.6
BA1000	70.12	90.21

TABLE I: Run Time

C. Tabulating Results:

In this section we summarize the results obtained for the aforementioned instances using the suggested methods. Four measures of interest are considered for each case, namely; The objective value, edge size (Number of edges remaining in the fragmented graph), Number of nodes in the largest cluster and the Number of components in the resulting graph. Each method is run for each instance 10 times and statistics are aggregated over the 10 runs. All the runs are for $k = 70$ Objective value is calculated using the formula given in [8], shown below:

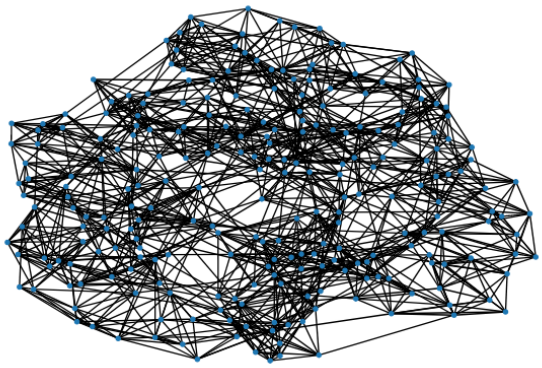
$$\frac{1}{2}\sum_{c\in C}\sigma_c * (1-\sigma_c) \tag{3}$$

[AP: Write note on different experimentations that you did.]

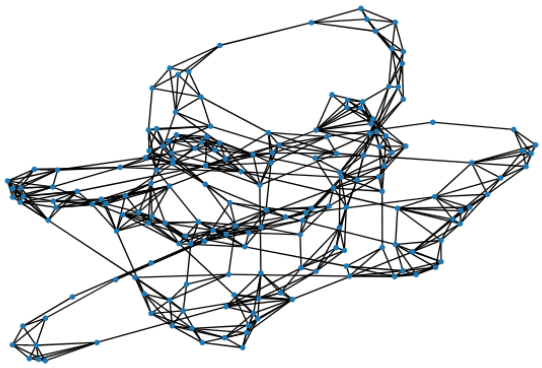
D. Sample Solution Graphs:

In this section, solution graphs are illustrated. Four instances from [8] are used to show solution graphs, namely; BA1000, FF250, WS250, ER250.

1) Instance name: Watts–Strogatz (WS250): The Watts-Strogratz graphs are highly dense and have a high clustering coefficient. This make them difficult to fragment. As seen in Figure 6, we see that the graph though, somewhat fragmented, is majorly still intact. $k = 70$ is still a relatively small number to significantly fragment the graph. The resulting graphs have an average fitness of 16110, which is comparable to results obtained in [8]. The largest clusters consist of on average, 180 nodes. See Figure 6.



(a) Watts–Strogatz (WS250)

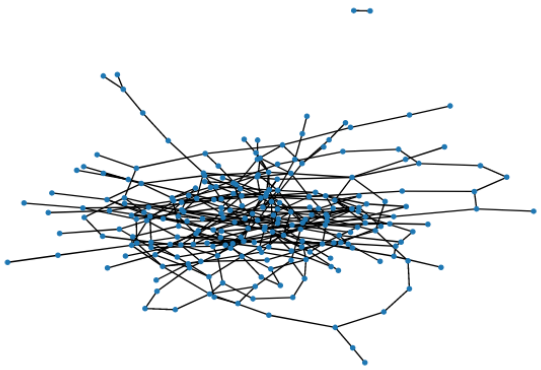


(b) Watts–Strogatz (WS250) result

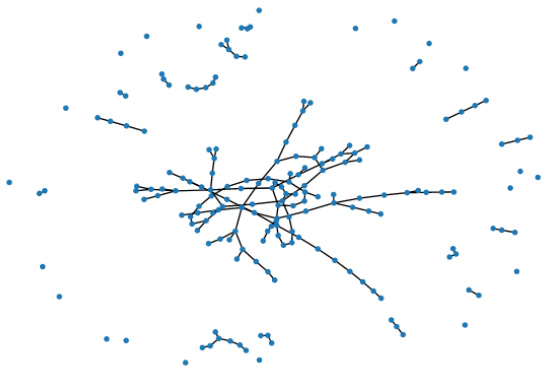
Fig. 6: Method 2 implementation on WS250 instance with $k = 70$; we see that the graph, though fragmented is still intact.

2) Instance name: Erdos–Reni (ER250): Method 1 is applied here, since it is noteworthy that Method 2, which employed flow rate computations, requires a non-fragmented input graph. We see that the output graph shown in 7b is significantly fragmented. The largest clusters on average have only about 28 nodes and have long path lengths which

augment the disruption of the network [4]. I believe these results are comparable to ones found in literature. See Figure 7.



(a) Erdos–Reni (ER250)



(b) Erdos–Reni (ER250) result

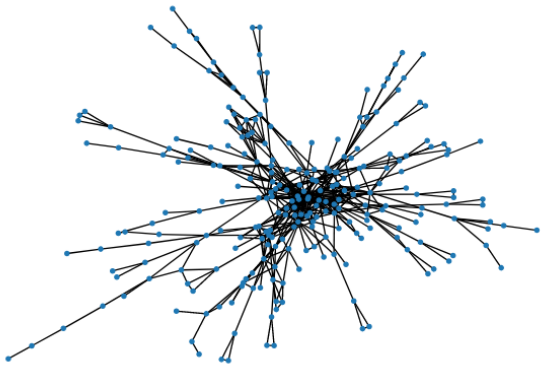
Fig. 7: Method 1 implementation on ER250 instance with $k = 50$; we see that the graph is significantly fragmented

3) Instance name: Forest Fire (FF250): Method 2, for the Forest Fire (FF250) instance where $k = 50$ and $k = 70$ significant nodes are identified and removed. We see that the network is significantly fragmented. We find that on average, the largest clusters have just 12 nodes as reported in table IV. See Figure 8.

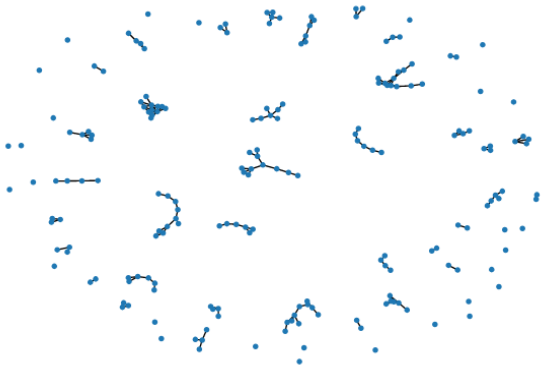
4) Instance name: Barabasi–Albert (BA1000): Method 2, for the Barabasi–Albert (BA1000) instance where $k = 50$ and $k = 70$ significant nodes are identified and removed. We see that the network is significantly fragmented. On average. the largest clusters have around 12 nodes. See Figure 9.

IV. Discussion:

The two methods seem to have results comparable, at lease for most instances, to [8].
A quick note on comparison of the two methods: I find that method 1 is better at creating sub graphs with lower objective values and lower number of total edges, on average. However, Method 2 does better at creating sub graphs with larger number of components. A note of caution, since we have effectively



(a) Forest Fire (FF250)



(b) Forest Fire (FF250) result $k = 50$

(c) Forest Fire (FF250) result $k = 70$

Fig. 8: Method 2 implementation on FF250 instance with $k = 50$ and $k = 50$; we see that the graph is significantly fragmented

only three instances to compare the two methods with, I refrain from commenting on if one is better than the other, or dealing with that in this work.

V. Conclusion:

This work proposed Ant-Colony algorithm and ACO with Breadth-First search approaches for attaining solutions to the

Instance	Method 1				Method 2			
	μ	σ	min	max	μ	σ	min	max
WS250	16110	0	16110	16110	16110	0	16110	16110
ER250	670.6	265.4	383	1274	NA	NA	NA	NA
FF250	225.2	33.23	168	264	302.2	4.19	295	312
BA1000	728.8	20.98	704	751	3376.4	7.09	3372	3389

TABLE II: Objective Values

Instance	Method 1				Method 2			
	μ	σ	min	max	μ	σ	min	max
WS250	594	5.03	587	600	601.2	4.93	593	608
ER250	101.1	3.634	97	109	NA	NA	NA	NA
FF250	105.4	3.94	99	111	134.4	1.3498	131	136
BA1000	353.8	1.308	352	355	555.7	0.836	556	558

TABLE III: Number of Edges

Instance	Method 1				Method 2			
	μ	σ	min	max	μ	σ	min	max
WS250	180	0	180	180	180	0	180	180
ER250	27.9	10.53	13	47	NA	NA	NA	NA
FF250	12	2.94	8	15	10	1.8	7	12
BA1000	11.8	0.4472	11	12	33	0	33	33

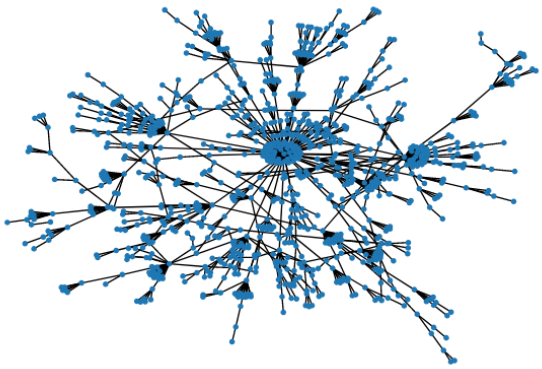
TABLE IV: Size of Largest Cluster

Instance	Method 1				Method 2			
	μ	σ	min	max	μ	σ	min	max
WS250	1	0	1	1	1	0	1	1
ER250	64	3.77	56	69	NA	NA	NA	NA
FF250	89.5	1.957	86	93	72.1	1.16	71	75
BA1000	576.2	1.3038	575	578	372.8	0.83	372	374

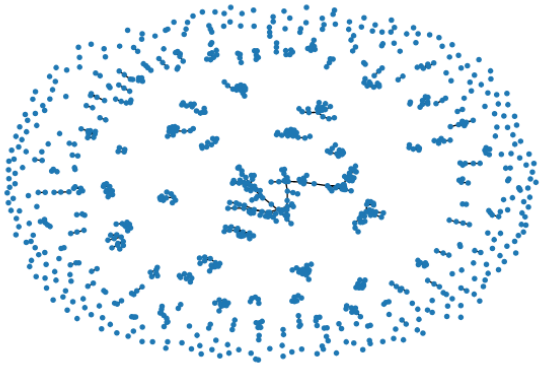
TABLE V: Number of Components

critical node detection problem. As far as I know, neither method had been previously applied specifically for CNP. Ant-Colony based techniques are known to be robust to change and are computationally inexpensive.

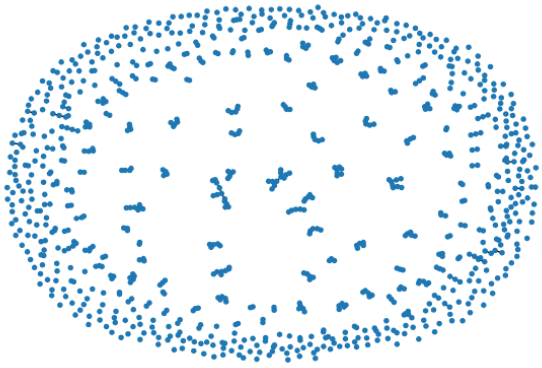
For assessing the quality of the algorithms, 4 benchmark problems from [8] were used. These problems are large problems with high number of nodes. We find that the algorithm generates good results for the benchmark problems are computes them in reasonable time. I believe that Ant-Colony optimization is suited for such problem statement and yield good results. Future work include testing on more benchmark problems, possibly coming up with a pheromone depositing rule which is a function of path length (to generate sub-graphs with large paths), possibly using ACO with other local-search algorithms.



(a) Barabasi–Albert (BA1000)



(b) Barabasi–Albert (BA1000) result $k = 50$



(c) Barabasi–Albert (BA1000) result $k = 70$

Fig. 9: Method 2 implementation on BA1000 instance with $k = 50$ and $k = 70$; we see that the graph is significantly fragmented

REFERENCES

[1] Roberto Aringhieri, Andrea Grosso, and Pierre Hosteins. A genetic algorithm for a class of critical node problems. *Electronic Notes in Discrete Mathematics*, 52:359–366, 2016.

[2] Ashwin Arulselvan, Clayton W Commander, Lily Elefteriadou, and Panos M Pardalos. Detecting critical nodes in sparse graphs. *Computers & Operations Research*, 36 (7):2193–2200, 2009.

[3] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[4] Stephen P Borgatti. Identifying sets of key players in a social network. *Computational & Mathematical Organization Theory*, 12(1):21–34, 2006.

[5] Thang N Dinh and My T Thai. Precise structural vulnerability assessment via mathematical programming. In *2011-MILCOM 2011 Military Communications Conference*, pages 1351–1356. IEEE, 2011.

[6] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[7] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[8] Mario Ventresca. Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem. *Computers & Operations Research*, 39(11):2763–2775, 2012.

[9] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684): 440, 1998.

[10] Wikipedia. Watts–strogatz model the free encyclopedia, 2019.

[11] Richard Wollmer. Removing arcs from a network. *Operations Research*, 12(6):934–940, 1964.

[12] Jinhui Yang, Xiaohu Shi, Maurizio Marchese, and Yanchun Liang. An ant colony optimization method for generalized tsp problem. *Progress in Natural Science*, 18(11):1417–1422, 2008.