Efficient heuristic algorithm for identifying critical nodes in planar networks

Dalaijargal Purevsuren [a,b] , Gang Cui [a]

[a] School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China
[b] Department of Information and Computer Science, School of Engineering and Applied Sciences, National University of Mongolia, Ulaanbaatar 14201, Mongolia
dalaijargal@gmail.com
Version 1 (July 24, 2018)

Abstract: This paper presents a new method for identifying critical nodes in planar networks. We propose an efficient method to evaluate the quality of solutions by using special properties of planar networks. It enables us to develop a computationally efficient heuristic algorithm for the problem. The proposed algorithm assisted on randomly generated planar networks. The experimental results are reported.

Key words: Critical node detection, Planar networks, Heuristics, NP-complete, Combinatorial optimization.

1. Introduction

Items in our world are connected with relationships. The items can be animals, machines, chemical elements, organizations and etc. These items and relationships can be represented as a network. Discovering knowledge from the networks (network mining) is an important task for many practical cases including drug designing, social network analysis, urban planning, and so on. Identifying a set of critical (or key) nodes in networks is a special case problem of network mining. A set of nodes is called critical if their failure causes damage in network connectivity. Given a constraint, maximizing the damage in network connectivity is called the critical node detection problem. It has several variants. In this study, we only consider the variant whose damage in network is measured by pairwise connectivity in the remaining network, and here the constraint is an upper limit on the number of critical nodes to be removed (it is called budget). The variant is called the CNP. The mathematical formulation is given as follows. Given a graph $G(V, E)$, and the number of nodes to be removed k, the CNP is formulated an optimization problem with equation (1) where S is a set of nodes to be removed from $G(V, E)$, C is the set of all components in the remaining graph $G[V \backslash S]$, c_i is the size of ith element in C. The CNP is known to be NP-complete [1] on general graphs.

$$F(S) = \min \sum_{i \in C} \frac{c_i(c_i-1)}{2} \quad (1)$$

$$Subject\ to:\ |S| \leq k \quad (2)$$

There are a number of heuristic approaches having local search (LS) for the CNP [1], [2], [3], [9] that LS plays main role. The LS-based heuristic approaches need to evaluate the quality of solution at each move. In order to evaluate the solution of the CNP, it is necessary to find connected components that requires O(max{$|V|$, $|E|$}) computational time [4] where $|V|$ is the number of nodes and $|E|$ is the number of edges of the graph. Calling the solution evaluation at each move makes the LS-based heuristic approaches very slow, and therefore, the problem size for these approaches is often limited by several thousands, e.g., up to five thousand nodes. There are a few greedy-based heuristic approaches [5], [6]. The approach described in [5] is designed for larger networks. With growing data availability, networks are also growing in size, and therefore there is a growing demand to develop computationally efficient approaches for larger networks.

In this paper, we propose a method that detects critical nodes in networks having up to a hundred thousand nodes under a requirement of planarity of networks. Our contributions are: (1) we transform the original CNP objective into two sub-objectives (maximizing the number of components; minimizing the total square difference in component size). (2) We propose a mechanism that optimizes the two sub-objectives simultaneously by investigating the right component size. It enables us to develop more efficient algorithms and consider bigger network instances.

The paper is organized as follows. Section two presents a new method for evaluating the quality of solutions. Section three describes the proposed method in detail. Two planar network sets are proposed as benchmark test problems in the section four. Section five presents the results of computational experiment. The conclusion and possible future extension are given in the section six.

2. Solution evaluation method

In this section, we introduce a new way to evaluate the quality of solutions for the CNP. To do this, we consider two sub-objectives: maximizing the number of components in the remaining graph; minimizing the total square difference in

component size. Let n be the number of components in the graph. The new way for solution evaluation can be represented as bi-objective optimization problem as follows where S is a set of nodes to be removed from $G(V,E)$, C is the set of all components in the remaining graph $G[V\backslash S]$ and c_i is the size of ith element in C.

$$H(S) = \{h_1(S),\ h_2(S)\} \quad (3)$$

$$\max\ \ h_1(S) = n; \quad (4)$$

$$\min\ \ h_2(S) = \sum_{i=1}^{n}\left(c_i - \frac{1}{n}\sum_{i=1}^{n}(c_i)\right)^2; \quad (5)$$

$$subject\ to\colon\ |S| \le k; \quad (6)$$

Lemma 1. *Let A be a partition of $G = (V,\ E)$ into n components obtained by deleting a set S of nodes, where $|\ S\ | = k$. Then the objective function $F(S) \ge n * \frac{\binom{|V|-k}{n} * \left(\binom{|V|-k}{n}-1\right)}{2}$ with equality holding if and only if $\omega_i = \omega_j, \forall i,j \in A$ where ω_i is the size of ith component of A.* [1]

Lemma 2. *Let A and B be two sets of partitions obtained by deleting S_1 and S_2 sets of nodes, respectively, from graph $G = (V,E)$, where $|S_1|=|S_2|= k$. Let n_1 and n_2 be the number components in A and B, respectively, and $n_1 \ge n_2$. If $\omega_i = \omega_j, \forall i,j \in A$, then we obtain a better objective function value by deleting the set S_1.* [1]

These two lemmas examine partitions having absolute equality condition on component sizes. An other words, when one objective equals to zero, i.e., $h_2(S) = 0$ that is the global optimal for $h_2(S)$, they examine the impact of another objective on the quality of solution. We introduce a lemma that examines both objectives' changes, not requiring an objective to be fixed to a value.

Lemma 3. Let A and B be the partitions obtained by removing sets of critical nodes S_1, S_2, respectively, where $|S_1|=|S_2|= k$. If the following two conditions holds $h_1(S_1) \ge h_1(S_2)$ and $h_2(S_1) \le h_2(S_2)$, it implies that S_1 is better solution than S_2, i.e., $F(S_1) \le F(S_2)$.

According to Lemma 3, function $H(S)$ can be considered as function for solution evaluation instead of using $F(S)$ directly. We also propose a mechanism that optimize $h_1(S)$ and $h_2(S)$ function simultaneously under a requirement of $|S| \le k$. Now let us consider following lemma with an assumption.

Lemma 4: Assume that a given component size γ, we can divide the graph into equally γ-sized components. Let γ_1, γ_2 be any given two component sizes. If they hold $\gamma_1 \le \gamma_2$, it implies that $h_1(S_1) \ge h_1(S_2)$ where S_1, S_2 are the sets of nodes to be removed with γ_1, γ_2, respectively, and $|S_1| = |S_2| = k$.

From lemma 4, decrease in component size γ will imply increase in $h_1(S)$. This means that component size γ has control on $h_1(S)$. If we find out a relation between component size γ and $h_2(S)$, then optimizing component size γ will lead to optimizing $H(S)$. In addition to optimize $H(S)$, we should be very careful to the constraint on the number of critical nodes. To reach these goals simultaneously, we propose two simple procedures, m-divider and γ-cutter, that optimizes $h_2(S)$ as considering the constraint.

3. Planar CNP solver

In this section, we introduce a heuristic algorithm for detecting critical nodes in planar graphs / networks (called planar_cnp). At first, we give facts on planar graphs and definitions that are used for explaining the algorithm.

Fact 1. A planar graph can be represented as a breadth first search (BFS) tree.

Fact 2. Deletion of a level in BFS tree of planar graphs divides the planar graphs into at least two parts.

Definition 1. A set of nodes that divides a BFS tree into m parts will be called m-divider such that the height of parts is equal with respect to BFS tree.

Definition 2. Given a BFS tree, and an integer γ, if a level of the BFS tree holds following two conditions, we call them γ-level.

(1) It cuts a piece from the tree such that the number of nodes in the piece is in an interval of $[w \cdot \gamma, \gamma]$ where w is an algorithmic parameter from (0, 1) interval,
(2) The number of nodes in the level is the smallest among the levels satisfying condition 1.

Definition 3. Given an BFS tree, and an integer γ, a collection of γ-level obtained by the following 3 steps is called γ-cutter.

(1) Examine levels starting from the root.
(2) If a level satisfies the conditions for γ-level, store it, and make it root.
(3) Repeat these steps until reaching leaves.

Lemma 5: The CNP is NP-hard on planar graphs.

The proposed heuristic has two main phases: (1) pre-processing for normalizing input networks (called Pre-processing phase), (2) heuristically constructing a near-optimal solution (called Constructor phase).

3.1 Pre-processing phase

The goal of pre-processing phase is to divide very dense networks (e.g., having small diameter) into smaller and sparser parts. We use a modified version of the FCS algorithm proposed in [8]. The FCS algorithm examines all fundamental cycle separators and selects a cycle that divides the graph into two balanced components. For more information about fundamental cycle separator, the interested readers are referred to [10], [8]. We modify it by selecting two cycles that divides the graph into three components such that as possible as balanced in size. We call it 3FCS. The algorithm is described in Algorithm 1 where $c.widest$ indicates the widest level size in the BFS tree of component c. We denote the algorithm by "$FCS\_divider$". The function $create\_bfs\_tree(G)$ in Algorithm 1 and 3 creates a BFS tree for each connected component in graph G, and returns the set of BFS tree T and total height H. To determine the root of BFS tree, the function calls height maximization method proposed in [8] that finds a root having a taller BFS tree.

Algorithm 1. $FCS\_divider[G, k]$

$s_{pre} \leftarrow \emptyset$;
Repeat
  $isDone \leftarrow true$;
  $[T, H] \leftarrow create\_bfs\_tree(G[V \backslash s_{pre}])$;
  For each c *in* T **do**
    If $((c.widest > p_0)\&\&(|c| > \sqrt{|G|}))$ //p_0 *is a parameter. The default value is* $2\sqrt{|c|}$.
      $s \leftarrow find\ 3FCS(c)$;
      $s_{pre} \leftarrow s_{pre} \cup s$;
      $isDone \leftarrow false$;
    End-if
  End-for
Until $(isDone == true)$;
Return s_{pre};

3.2 Constructor phase

From Lemma 3, we can make a potential conclusion. There will be a trend of improvement in objective function values by optimizing the following two sub-goals: (1) maximizing the number of components in the remaining graph, (2) minimizing the total square difference in component size. This is the main underlying idea of constructor phase.

The $constructor$ defined in Algorithm 2 has two main components: $m\_divider$ and $cutter$ where s_{pre} is the result of pre-processing phase. The goal of $m\_divider$ is to divide the graph into a pre-defined number of parts with equal height with respect to BFS tree. The later, $cutter$, divides the resulting graph of $divider$ into smaller pieces. The $cutter$ heuristically investigates the component size γ. It is presented in Algorithm 3 in detail where $find\_gamma\_cutter$ is a function that returns γ-cutter for given γ, w and a BFS tree t.

Let γ be the ideal component size in the remaining graph. Let n be the number of components in the remaining graph. The component size γ has direct control on the number of components n according to lemma 4. If γ decreases, n increases, and vice versa. Finding the right γ value is vital to reach better objective value. The γ value is investigated heuristically over [1, k] interval. The $find\_gamma\_cutter$ procedure in algorithm 3 can be considered as a function that maps γ to n. The function is monotone on γ if there is no constraint on k. It enables us to use binary search instead of examining every value in the interval [1, k] to find the right γ value. In addition, the function works very economically on budget according to definition 2 and definition 3.

4. Benchmark problem instances

As far as we know, there are no publicly available datasets of planar networks. In order to evaluate the quality of the proposed algorithm, we prepare two types of benchmark dataset. They are generated randomly with Mathematica software.

4.1 Grid with Random Removal (GRR) dataset

The number of graphs is six with sizes of {500; 1,000; 5,000; 10,000; 50,000 and 100,000}. The generation method has two steps as follows. In the first step, the grid graph is created with GridGraph function. In the second step, we select

a node at random, find its neighbor nodes, and remove the selected node and its neighbors. Obviously, all incident edges are also removed. The amount of removed nodes is around 5% of total nodes in original grid graph. An illustration of the networks is given in Figure 1. For further information, please look at https://www.wolfram.com/mathematica/new-in-8/graph-and-network-analysis/subtract-random-neighborhoods.html.

Algorithm 2. $constructor(G, k, s_{pre}, w)$

$m \leftarrow 0$;
$s^* \leftarrow \emptyset$;
$num\_no\_imp \leftarrow 0$;
Repeat
    $m \leftarrow m + 1$;
    $s_1 \leftarrow m\_divider(G, m, s_{pre})$;
    if $(|s_1| > k)$ **then**
        $break$;
    end
    $s_2 \leftarrow cutter(G, k, s_1, w)$;
    if $(F(s^*) > F(s_2))$ **then**
        $s^* \leftarrow s_2$;
        $num\_no\_imp \leftarrow 0$;
    else
        $num\_no\_imp ++$;
    end
Until $\big((m \geq k)$ && $(num\_no\_imp \geq p_1)\big)$; // p_1 is a parameter. The default value is 10.
Return s^*

Algorithm 3. $cutter(G, k, s_1, w)$

$[T, H] \leftarrow create\_bfs\_tree(G[V \setminus s_1])$;
$a \leftarrow 1$;
$b \leftarrow k$;
$s^* \leftarrow \emptyset$;
Repeat
    $m \leftarrow \frac{a+b}{2}$;
    $\gamma \leftarrow m$;
    $s_2 \leftarrow s_1$;
    For each component t in T **do**
        $s \leftarrow find\_gamma\_cutter(\gamma, w, t)$; // w is a parameter. The default value is 0.5.
        $s_2 \leftarrow s_2 \cup s$;
    End
    If $(|s_2| \leq k)$ **then**
        If $(F(s^*) > F(s_2))$ **then**
            $s^* \leftarrow s_2$;
        End
        $b \leftarrow m$;
    Else
        $a \leftarrow m$;
    End
Until $(b - a > 1)$;
Return s^*;

4.2 Delaunay Triangulation (DT) dataset

The number of graphs is six with sizes of {500; 1,000; 5,000; 10,000; 50,000 and 100,000}. The generation method has two steps. In the first, sequence of random points in 2d are generated with RandomReal[{0,1},{n,2}] function where n is the number of nodes to be created. Delaunay triangulation on these points is created with DelaunayTriangulation[pts] in the second step where pts is a vector of the result of RandomReal function. An illustration of the networks is given in Figure 2. For further information, please look at http://reference.wolfram.com/language/ComputationalGeometry/ref-/DelaunayTriangulation.html.

The number of critical nodes (budget) is around 5 and 10 percent of total nodes for each network we generate. Therefore, we have totally 24 benchmark instances. Some characteristics of above-mentioned two datasets are summarized in Table 1. More planar network instances will be generated in the future works.

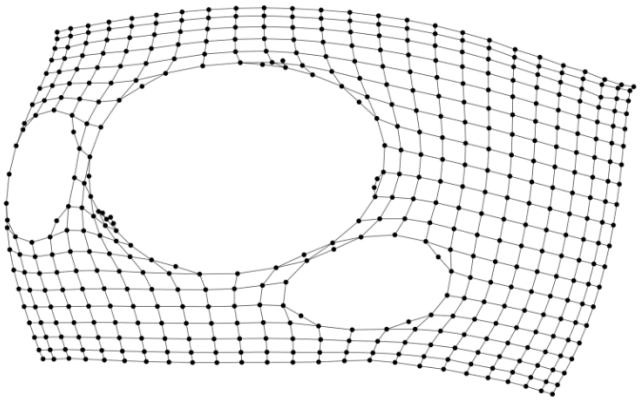

Figure 1. An illustration of GRR dataset

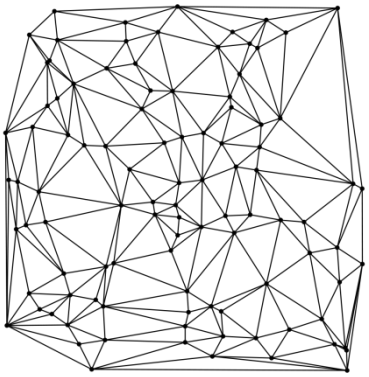

Figure 2. An illustration of DT dataset

Table 1. Some characteristics including number of nodes and edges, the critical nodes *k* for the datasets

Type	**Problem name**	**# of nodes**	**# of critical nodes k (5%; 10% of |V|)**	**# of edges**
DT	dt500	500	{25; 50}	1,480
	dt1000	1000	{50; 100}	2,982
	dt5000	5,000	{250; 500}	14,976
	dt10000	10,000	{500; 1,000}	29,973
	dt50000	50,000	{2,500; 5,000}	149,975
	dt100000	100,000	{5,000; 10,000}	299,959
GRR	grr_20_25	475	{25; 50}	877
	grr_25_40	956	{50; 100}	1,799
	grr_50_100	4,760	{250; 500}	9,104
	grr_100_100	9,510	{500; 1,000}	18,268
	grr_200_250	47,590	{2,500; 5,000}	92,066
	grr_250_400	95,373	{5,000; 10,000}	185,283

5. Computational Experiment

In this section, we assist the performance of proposed algorithm on the abovementioned 24 benchmark instances/ problems. The computational experiment consists of two sub-sections including algorithm configuration, and experimental result. The proposed algorithm was implemented in C++ and compiled with gcc 4.9.2. It was tested on a PC equipped with a 3.6GHz intel (R) core i7-7700 of CPU, and 8GB of RAM. Running time is limited to 100 seconds for all algorithms assisted in this work.

5.1 Algorithm configuration

The value for each parameter is presented in Table 2.

Table 2. Values for the parameters used for planar_cnp

Parameter	Value
p_0	$2\sqrt{\lvert C \rvert}$ [a]
p_1	10
w	0.5

[a] C is a component, $|C|$ is the size of component.

5.2 Experimental result

In this subsection, we examine the detailed results of planar_cnp. Summary results (mean, standard deviations, minimum, and maximum objective function value (OFV), average running time over 30 independent runs) for the planar_cnp algorithm are given in Table 3. An analysis on the number of nodes removed by planar_cnp is reported in Table 4. The number of average nodes removed by planar_cnp is less than the budget for all instances, especially for instances of DT dataset, it is less than 90 percent of the budget in total.

Table 3. Summary results (mean, standard deviation, minimum, maximum, average running time over 30 independent runs) for planar_cnp

Type	Problem name	# of critical nodes k	mean	s. d.	min	max	Avg. running time in second
DT	dt500	25	71,290.00	0.00	71,290	71,290	0.06
		50	39,633.53	7,365.22	35,314	58,236	0.06
	dt1000	50	306,466.00	0.00	306,466	306,466	0.09
		100	164,253.97	12,124.38	154,105	185,530	0.11
	dt5000	250	2,224,843.03	116,662.59	2,085,244	2,393,795	0.59
		500	1,052,627.10	38,046.54	985,809	1,139,024	0.75
	dt10000	500	7,777,632.07	621,001.52	6,216,701	8,376,072	1.30
		1,000	2,962,715.53	143,267.99	2,709,115	3,349,474	1.70
	dt50000	2,500	67,546,394.37	1,770,980.59	63,039,954	70,241,407	14.32
		5,000	14,808,784.27	234,396.76	14,201,336	15,204,668	22.32
	dt100000	5,000	145,125,342.90	4,561,779.66	136,156,323	154,477,441	47.27
		10,000	30,610,588.93	467,509.30	29,432,966	31,649,153	72.97
GRR	grr_20_25	25	54,716.33	15,505.00	48,868	112,575	0.01
		50	15,469.27	388.20	15,176	16,099	0.03
	grr_25_40	50	103,512.53	1,394.61	102,886	107,672	0.03
		100	36,769.80	1,936.93	34,694	38,619	0.04
	grr_50_100	250	809,367.37	9,296.30	798,848	821,810	0.13
		500	217,864.83	5,223.51	210,687	227,726	0.27
	grr_100_100	500	1,910,764.03	41,385.20	1,814,214	1,981,233	0.31
		1,000	474,838.10	7,457.23	446,811	487,721	0.62
	grr_200_250	2,500	11,112,083.23	74,638.10	10,926,700	11,224,455	3.50
		5,000	2,560,309.30	15,150.46	2,530,882	2,586,915	6.11
	grr_250_400	5,000	24,274,110.23	386,967.01	23,758,008	25,299,468	10.89
		10,000	5,548,277.40	109,323.85	5,276,661	5,662,673	20.96

Table 4. Number of nodes (average over 30 runs) removed by planar_cnp

Problem name	# of critical nodes k (budget)	Average nodes removed (*k*)		Problem name	# of critical nodes k (budget)	Average nodes removed (*k*)	
		by number	*by percent from budget*			*by number*	*by percent from budget*
dt500	25	24.0	96.0	grr_20_25	25	19.6	78.4
	50	44.4	88.8		50	49.0	97.9
dt1000	50	28.0	56.0	grr_25_40	50	47.4	94.9
	100	72.1	72.1		100	99.5	99.5
dt5000	250	229.9	91.9	grr_50_100	250	248.1	99.3
	500	460.9	92.2		500	496.7	99.3
dt10000	500	376.3	75.3	grr_100_100	500	495.2	99.0
	1,000	810.2	81.0		1,000	997.8	99.8
dt50000	2,500	2,031.0	81.2	grr_200_250	2,500	2,494.4	99.8
	5,000	4,545.1	90.9		5,000	4,987.6	99.8
dt100000	5,000	4,372.7	87.5	grr_250_400	5,000	4,988.5	99.8
	10,000	9,377.9	93.8		10,000	9,955.6	99.6
Total	24,975	22,372.5	89.6	Total	24,975	24,879.4	99.6

6. Conclusion

In this paper, an efficient method for evaluating solutions of the CNP is introduced. With the efficient method, we propose a novel algorithm for identifying the critical nodes in planar networks. The extensive computational experiment is conducted. For future work, we will consider quasi-planar graphs. Furthermore, the proposed method for solution evaluation is needed to investigate for applying it to more complex networks. Using other measures for damage in networks is also a possible next move.

Reference

[1] Arulselvan, A., Commander, C.W., Elefteriadou, L., Pardalos, P.M., 2009. Detecting critical nodes in sparse graphs. Computers & Operations Research 36, 2193–2200. https://doi.org/10.1016/j.cor.2008.08.016

[2] Aringhieri, R., Grosso, A., Hosteins, P., Scatamacchia, R., 2016. Local search metaheuristics for the critical node problem. Networks 67, 209–221. https://doi.org/10.1002/net.21671

[3] Purevsuren, D., Cui, G., Qu, M., Win, N.N.H., 2017. Hybridization of GRASP with Exterior Path Relinking for Identifying Critical Nodes in Graphs. IAENG International Journal of Computer Science 44, 157–165.

[4] Hopcroft, J., Tarjan, R., 1973. Algorithm 447: Efficient Algorithms for Graph Manipulation. Communications of the ACM 16, 372–378.

[5] Addis, B., Aringhieri, R., Grosso, A., Hosteins, P., 2016. Hybrid constructive heuristics for the critical node problem. Annals of Operations Research 238, 637–649. https://doi.org/10.1007/s10479-016-2110-y

[6] Pullan, W., 2015. Heuristic identification of critical nodes in sparse real-world graphs. Journal of Heuristics 21, 577–598. https://doi.org/10.1007/s10732-015-9290-5

[7] Garey, M.R., Jonhson, D.S., 1976. Some Simplified NP-Complete Graph Problems. Theoretical Computer Science 1 237–267.

[8] Holzer, M., Schulz, F., Wagner, D., Prasinos, G., Zaroliagis, C., 2009. Engineering planar separator algorithms. Journal of Experimental Algorithmics 14, 1.5. https://doi.org/10.1145/1498698.1571635

[9] Zhou, Y., Hao, J.-K., Glover, F., 2018. Memetic Search for Identifying Critical Nodes in Sparse Graphs. IEEE Transactions on Cybernetics 1–14. https://doi.org/10.1109/TCYB.2018.2848116

[10] Lipton, R.J., Tarjan, R.E., 1979. A Separator Theorem for Planar Graphs. SIAM Journal on Applied Mathematics 36, 177–189. https://doi.org/10.1137/0136016