Dynamically swapping vertices of the critical node problem

Xiang-Rong Zeng[1], Si-Yang Liu[2],Li-Fan Shen[1],Yong-Quan Chen[1],Zhang-Hua Fu[1*]

1. Robotics Laboratory for Logistics Service, Institute of Robotics and Intelligent Manufacturing, The Chinese University of Hong Kong, Shenzhen, 518172, China.
E-mail: zengxiangrong@cuhk.edu.cn, shenlifan@nyu.edu, ypchen@cuhk.edu.cn, fuzhanghua@cuhk.edu.cn

2. School of Software Engineering,Xi'an Jiao Tong University,Xi'an,710049,China
E-mail: liusiyang2016@stu.xjtu.edu.cn

Abstract: The Classic Critical Node Problem (CNP) is the problem about selecting k nodes in an undirected graph G=(V,E) and trying to make the number of connected point pairs in the residual graph which deletes the k nodes as small as possible. The problem has a wide range of applications in the areas of cyber security, disease control, biological analysis, social networking, and so on. Because this problem is an NP-hard problem, heuristic algorithm has become a common method for solving large-scale CNP problems. In this algorithm, node exchange operations are widely used. If an initial solution of a set of K nodes is given, $K \times (|V| - K)$ node exchange operations will need to be performed. The minimum complexity of existing algorithms to verify these switching operations is $O(|V| - K) \times (|V| + |E| + K \times D(G))$ (where $|V|$ is the number of nodes, $|E|$ Is the number of edges, D(G) is the maximum degree of the node in graph G). In this paper, a series of dynamic data structures are used to reduce the total time complexity of verifying the exchange operation of $K \times (|V| - K)$ vertices to $O(|V| - K) \times (|V| + |E|)$. This method is expected to significantly increase the efficiency of searching and improve the performance of algorithm.

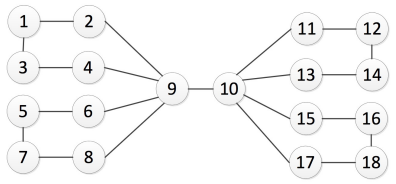
Key Words: Critical Node Problem; Node-Swap Operator; Dynamic Data Structure

1 Introduction

The Classic Critical Node Problem (CNP) is defined as follows: Given an undirected graph G=(V,E) (V is its point set, E is its edge set) and a positive integer K, it aims to find a set S consisting of K nodes, which would make the number of connected pairs of nodes in the residual graph G[V/S] as small as possible after deleting the nodes in the S set and their associated edges, i.e.:

$$Minimize \quad f(S) = \sum_{i=1}^{T} \frac{|c_i| + (|c_i|^{-1})}{2} \tag{1}$$

T is the total number of parts in the residual graph, and C_i represents the i-th part.



(a)

(b)

Fig. 1: (a) original graph (b) residual graph

The figure above shows an example of the CNP, where Figure 1.(a) is the original graph, including 18 nodes and 21 edges. When K = 2, the optimal solution S = 9, 10, and the corresponding residual graph is Figure 1. (b), the F (S) (the objective function value) of which is 24.

The CNP problem has been widely used in areas such as cybersecurity [1-2], disease control [3-4], biological analysis [5-6], and social network analysis [7-8]. For example, in social networks, nodes represent individuals, the edges represent the links between individuals and the CNP problem studies how to identify the key people in the social network.

CNP has been proved to be an NP-Hard problem [9], and is polynomially solvable only in some special cases [10-11]. When the scale of the problem is large, it is extremely difficult to find its optimal solution, so the heuristic algorithm has become a common solution. Most of the early heuristic algorithms were based on greedy rules [8,9,12] and local search [13-14] which is also based on greedy rules. In the existing heuristic algorithm, the node exchange operation (switching a pair of nodes which belong to S and V" S respectively is a widely used basic operation, and its computational complexity is one of the key factors affecting the efficiency of the algorithm. If an initial solution of a set of K nodes is given, $K \times (|V| - K)$ node exchange operations will need to be performed. The minimum complexity of existing algorithms to verify these switching operations is $O(|V| - K) \times (|V| + |E| + K \times D(G))$ (where D(G) is the maximum degree of the node in graph G) [14]. In worst cases, the complexity would be $O(|V| - K) \times (K \times |V| + |E|)$. In this paper, a series of dynamic data structures are used to reduce the total time complexity of verifying the exchange operation of $K \times (|V| - K)$ vertices to $O(|V| - K) \times (|V| + |E|)$. This method is expected to significantly increase the efficiency of searching and improve the performance of algorithm.

2 The Proposed Method

2.1 Algorithm process

Given the initial subset S and its corresponding residual graph $G[V \backslash S]$, exchange vertex v_i in $V \backslash S$ and v_j in S and calculate its object value. If the subset S has K nodes , the operation of swapping nodes between subset S and residual graph $G[V \backslash S]$ will be performed $K \times |V \backslash S|$ times. In this paper, the time complexity of the $K \times (|V| - K)$ times

vertex exchange operation is controlled within the range of $O(|V|-K)\times(|V|+|E|)$ through dynamic data structure. Our proposed method consists of two parts: 1. Record component information of each node of $G[V\backslash S]$ 2. Dynamically calculates the value of the object value after switching any pair of vertices. The calculation process is shown in algorithm 1, and the details are described in the following sections.

Algorithm 1 Dynamically evaluating solutions after swapping nodes of the CNP

Input: undirected graph $G=(V,E)$, subset S with K nodes, residual graph $G[V\backslash S]$
Output: objective value $f(S\cup\{v_i\}/\{vj\})$ after swapping each pair of nodes $v_i\in V\backslash S$ and $v_j\in S$

Prepare: Record component information of each node in $G[V\backslash S]$ // detailed in Section 2.2
// dynamically calculate the objective value after swapping any pair of nodes, see Section 2.3
for each node $v_i\in V\backslash S$ **do**
  Delete node v_i as well as its incident edges from $G[V\backslash S]$;
  Record component information of each node (if changed);
  $f(S\cup\{v_i\})\leftarrow$ Update objective after delete node $(V\backslash S,v_i)$;
  for each node $v_j\in S$ **do**
    for each node v_k adjacent to v_j in G **do**
      if v_k belongs to some component **then**
        $f(S\cup\{v_i\}/\{vj\})\leftarrow$ Update objective after connect node $(V/(S\cup\{v_i\}),v_j,v_k)$;
      end if
    end for
  end for
end for

2.2 Record component information of each node of G[V\S]

After deleting subset S and its associated edges from the original graph $G(V,E)$, the residual graph will be divided into several discrete components.Before switching vertices, do the following: An array of $Node\_Component$ with $|V|$ length is used to record the Component that each vertex belongs to in the residual graph $G[V\backslash S]$. To do this, we can set the initial value of each element in the array $Node\_Component$ to Null (if the corresponding vertex is S) or 0 (otherwise), and the initial value of the number i and the component number $Current\_Component$ to 1. Then, traverse the array $Node\_Component$ sequentially and perform the following actions :

(1) . If $Node\_Component[i]$ is Null or greater than 0 ,$i=i+1$.

(2) . If $Node\_Component[i]$ equals 0, a depth-first search (DFS) is performed from vertex i, setting the $Node\_Component$ value of the traversed vertex to $Current\_Component$. After DFS , $i=i+1$ and $Current\_Component=Current\_Component+1$.

Loop through the above process until each vertex in the residual graph is traversed . Now, the component information that each vertex belongs to is recorded in the array $Node\_Component$.

Finally, traverse the array $Node\_Component$ again to get the number of vertices in each component and record it in the array ComponentSize. The length of the array Component Size is the number of components in the residual graph.

2.3 Dynamic switching vertex

Based on the above data structure, dynamic vertex exchange can be performed. Given the initial subset S, the process of dynamic exchange between each vertex v_i in $V\backslash S$ and each vertex in S mainly includes the following two steps:

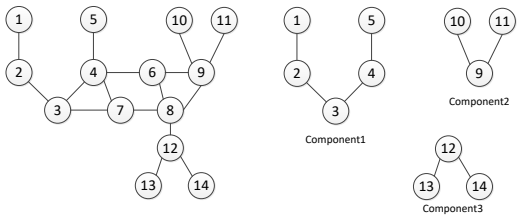
Step 1 : Delete node V_i from $V\backslash S$ and add node V_i to S : Remove vertex v_i and associated edges from residual graph $G[V\backslash S]$. During this process, the component that v_i belongs to will change (the number of vertices will be reduced by 1, and it is possible to split the component into multiple $Sub\_ComponentsSub\_Component$), and the others components will not change (array If ComponentChanged is used to record whether a Component has changed). Perform a DFS on the changed Component, and array Node$Sub_Component$ to record the $Sub\_Component$ information of the changed component, and array $Sub_Component$Size to record the number of vertices in each $Sub\_Component$ (method is similar as section 2.2). After the above operation, the object value $f(S)$ can be calculated based on the number of vertices of each Component and $Sub\_Component$.

It should be emphasized that for each vertex v_i in $V\backslash S$, it only needs to perform the above procedure only once when it is exchanged with each vertex in S.

Step 2 : Delete node $V_j\neq V_i$ from S and add V_j to $V\backslash S$: After adding vertex v_j to the last residual graph, it is possible to connect multiple Components or $Sub_Component$s to a new Component. The object value will increase..Therefore, for each adjacency node v_k of v_j in $G(V,E)$, v_k can be determined whether v_k belongs to a Component or $Sub\_Component$ based on the comprehensive information of array $Node\_Component$, array IfComponentChanged , and array $Node\_Sub\_Component$. If Vk belongs to a Component or $Sub\_Component$, query the array ComponentSize and array $Sub\_Component\_Size$ to get the number of vertices.Determine which components or subcomponents can be connected by v_k, and then calculate the added value of the target function (the complexity is $O(1)$).

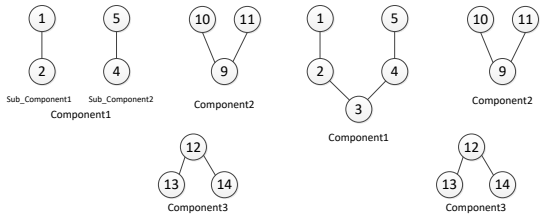
Repeat the above process until all v_j's adjacency points are traversed in the original graph $G(V,E)$. The object value after switching vertices v_i in $V\backslash S$ and v_j in s can be calculated.

2.4 Example



(a) the original graph

(b) the initial solution with S= 6,7,8



(c) graph after deleting node 3

(d) graph after swapping node 3 and 6

(e) graph after swapping node 3 and 7

(f) graph after swapping node 3 and 8

Fig. 2: An example of vertex-swap using our proposed method

Figure 2 is an instance of CNP, where figure 2.(a) is the original graph, including 14 vertices and 16 edges. Fig. 2.(b) is an initial solution when $K = 3(S = 6, 7, 8)$, whose object value $f(S) = 16$.The residual graph is divided into three discrete Components, including 5, 3, and 3 vertices (as shown in the table below).

Table 1: Node_Component Array

Node	1	2	3	4	5	6	7
Component	1	1	1	1	1	null	null
Node	8	9	10	11	12	13	14
Component	null	2	2	2	3	3	3

Table 2: Component_Size Array

Component	1	2	3
Size	5	3	3

Figure 2.(c) is the residual graph obtained after vertex 3 is deleted based on figure 2.(b), where $Component1$ changes and is further divided into $Sub\_Component1$ and $Sub\_Component2$. Each SubComponent contains 2 vertices.The following tables are the relevant information, and the corresponding object value is 8.

Table 3: If_Component_Changed Array

Component	1	2	3
If Changed	true	false	false

Table 4: Node_Sub_Component Array

Node	1	2	4	5
Sub_Component	1	1	2	2

Table 5: Sub_Component_Size Array

Sub_Component	1	2
Size	2	2

$Sub\_Component$

Based on the above data structure, if the vertex 6 in S is added in residual graph (equivalent to exchange vertex 3 and 6), traversing the vertex 6 adjacent vertexes $(4, 8, 9)$, and perform the following steps:

(1) Check vertex 4. According to the array $Node\_Component$, vertex 4 belongs to Component 1 in $G[V\backslash S]$. Furthermore, according to the array $If\_Component\_Changed$, $Component1$ changes after vertex 3 is deleted. The vertex 4 belongs to $Sub\_Component2$ (including 2 vertices).Therefore, vertex 6 will be connected with $Sub\_Component2$ to form a new component has 3 vertices, and the value of the object value will increase by $3 \times (3-1)/2 - 2 \times (2-1)/2 = 2$, that is, from 8 to 10.

(2) Continue checking vertex 8, because it belongs to initial subset S, skip it directly;

(3)Continue to check vertex 9, which belongs to $Component2$ (containing 3 vertices) in $G[V\backslash S]$ and is unchanged after vertex 3 is deleted.Therefore, through vertex 9, two Components containing 3 vertices can be connected to a new component containing 6 vertices, and the object value will be increased by $6 \times (6-1)/2 - 3 \times (3-1)/2 - 3 \times (3-1)/2 = 9$, that is, from 10 to 19 (Fig.2.d).

Similarly, it can be calculated that the object value of switching vertex 3 and vertex 7 is 10 (figure 2.(e)), and the object value of switching vertex 3 and vertex 8 is 23 (figure 2.(f)).Switching operations for other vertices in V/S are similar and will not be repeated here.

3 Complexity analysis

(1) As mentioned in section 2.2, before switching vertices, the Depth-First-Search is required to obtain the component information of $G[V\backslash S]$, whose computational complexity is $O(|V| + |E|)$.

(2) As described in section 2.3, for each vertex vi in V/S, perfom a Depth-First-Search to update the Component and $Sub\_Component$ information after v_i is deleted and the object value will be recalculated. In the worst case, the complexity is $O(|V| + |E|)$.Because $V\backslash S$ has a total of $|V| - K$ vertices, the total complexity of the above operation will not exceed $O(|V| - k) \times (|V| + |E|)$.

(3) As described in section 2.3, after vertex vi in $V\backslash S$ is deleted from the residual graph, for each vertex v_j in s, it is necessary to traverse its adjacency node, determine which component or $Sub\_Component$ each adjacency node belongs to, and update the object value in real time.Based on the data structure described in sections 2.2 and 2.3, the operating complexity for each adjacency node is $O(1)$.Therefore, after removing vertex v_i in $V\backslash S$, in turn , add v_j in S to residual. The total complexity of add each vertex in S to residual graph is $O(\sum_{i=1}^{k} Degree(v_i)) \leq O(|E|)$.Because $V\backslash S$ has a total of $|V| - K$ vertices, the total complexity of the above operation is $O(|V| - k) \times |E|)$.

(4) To sum up, given an undirected graph $G(V, E)$ and the

initial subset $S(|S| = K)$, dynamic exchange of $K \times (|V| - K)$ pairs of the vertexes does not exceed the total complexity of $O(|V| + |E|) + O((|V| - K) \times (|V| + |E|)) + O((|V| - K) \times |E|) = O((|V| - K) \times (|V| + |E|))$.

4 Conclusions

The Critical Node Problem(CNP) is an important combinatorial optimization Problem with a wide application background.In the heuristic algorithm for solving this problem, vertexes exchange operation is a widely used basic operation.Given an undirected graph $G(V, E)$ and the initial subset $S(|S| = K)$, there are $K \times (|V| - K)$ switching vertexes operations.The minimum complexity of the above exchange operation is $O(|V| - k) \times O(|V| + |E| + K \times D(G))$, where $D(G)$ is the maximum degree of each vertex in the original graph.In this paper , we test $K \times (|V| - K)$ vertices exchange operation complexity is $O((|V| - K) \times (|V| + |E|))$ by using a series of dynamic data structure.In the future, we will combine this technology with advanced search methods to design high performance CNP algorithm.

Acknowledgments

This paper is partially supported by the National Natural Science Foundation of China (grant No: U1613216), the State Joint Engineering Lab on Robotics and Intelligent Manufacturing, and Shenzhen Engineering Lab on Robotics and Intelligent Manufacturing, from Shenzhen Gov, China.

References

[1] Lozano M, Garca-Martnez C, Rodrguez F J, et al. Optimizing network attacks by artificial bee colony[J]. Information Sciences, 2017, 377: 30-50.

[2] Shen Y, Nguyen N P, Xuan Y, et al. On the discovery of critical links and nodes for assessing network vulnerability[J]. IEEE/ACM Transactions on Networking (TON), 2013, 21(3): 963-973.

[3] Kuhlman C J, Kumar V S A, Marathe M V, et al. Finding critical nodes for inhibiting diffusion of complex contagions in social networks[C]//Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Berlin, Heidelberg, 2010: 111-127.

[4] Ventresca M, Aleman D. A. derandomized approximation algorithm for the critical node detection problem[J]. Computers & Operations Research, 2014, 43: 261-270.

[5] Boginski V, Commander C W. Identifying critical nodes in protein-protein interaction networks[M]//Clustering challenges in biological networks. 2009: 153-167.

[6] Tomaino V, Arulselvan A, Veltri P, et al. Studying connectivity properties in human protein?protein interaction network in cancer pathway[M]//Data Mining for Biomarker Discovery. Springer, Boston, MA, 2012: 187-197.

[7] Borgatti S P. Identifying sets of key players in a social network[J]. Computational & Mathematical Organization Theory, 2006, 12(1): 21-34.

[8] Ventresca M, Aleman D. Efficiently identifying critical nodes in large complex networks[J]. Computational Social Networks, 2015, 2(1): 6.

[9] Arulselvan A, Commander C W, Elefteriadou L, et al. Detecting critical nodes in sparse graphs[J]. Computers & Operations Research, 2009, 36(7): 2193-2200.

[10] Addis B, Di Summa M, Grosso A. Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth[J]. Discrete Applied Mathematics, 2013, 161(16-17): 2349-2360.

[11] Shen S, Smith J C. Polynomial?time algorithms for solving a class of critical node problems on trees and series?parallel graphs[J]. Networks, 2012, 60(2): 103-119.

[12] Pullan W. Heuristic identification of critical nodes in sparse real-world graphs[J]. Journal of Heuristics, 2015, 21(5): 577-598.

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

[14] Aringhieri R, Grosso A, Hosteins P, et al. Local search metaheuristics for the critical node problem[J]. networks, 2016, 67(3): 209-221.