A fast heuristic algorithm for the critical node problem

Yangming Zhou
Université d'Angers
2 Boulevard Lavoisier
49045 Angers cedex 01, France
zhou.yangming@yahoo.com

Jin-Kao Hao*
Université d'Angers
2 Boulevard Lavoisier
49045 Angers cedex 01, France
jin-kao.hao@univ-angers.fr

ABSTRACT

The critical node problem (CNP) aims to identify a subset of critical nodes in an undirected graph such that removing these critical nodes minimizes the pairwise node connectivity over the residual graph. CNP has various applications; however, it is computationally challenging. This paper introduces FastCNP, a fast heuristic algorithm for solving the problem. FastCNP employs an effective two-phase node exchange strategy to locate high-quality solutions and applies a destructive-constructive perturbation procedure to drive the search to new regions when the search stagnates. Computational results on 16 popular benchmark instances show that FastCNP finds improved best results (new upper bounds) for 6 instances, and matches the best-known results for 9 instances.

KEYWORDS

Critical node problem; heuristics; iterated local search; combinatorial optimization.

ACM Reference format:
Yangming Zhou and Jin-Kao Hao. 2017. A fast heuristic algorithm for the critical node problem. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3067695.3075993

1 INTRODUCTION

Given an undirected graph $G = (V, E)$ and an integer K, the critical node problem (CNP) is to identify and extract a subset of nodes $S \subset V$ (where $|S| \leqslant K$) from G in order to minimize the total number of node pairs still connected in the connected components of the residual graph $G[V \setminus S]$ [3].

$$\min \quad f(S) = \sum_{i=1}^{n_c} \binom{|C_i|}{2} \tag{1}$$

$$\text{s.t.} \quad |S| \leqslant K, \tag{2}$$

where n_c is the number of connected components in the residual graph, and $|C_i|$ represents the number of nodes in the connected component C_i. The nodes in S are called *critical nodes* because their removal results in minimization of the above objective function.

CNP is known to be NP-hard [3] in the general case even if there are polynomially solvable special cases [5]. The computational challenge and wide range of applications of CNP have motivated a variety of solution approaches in the literature, including both exact [3, 6] and heuristic algorithms [1–4].

In this study, we design and implement a fast heuristic algorithm for CNP called FastCNP. The main originality behind FastCNP is its low-complexity "two-phase node exchange" operation, which breaks the traditional high-complexity "node swap" operation into two distinct and constrained "single node move" operations. Based on the two-phase node exchange strategy, we create a fast neighborhood search algorithm which is able to locate high-quality solutions. In addition, we design a destructive-constructive perturbation procedure which enables the search to escape from local optima traps and visit new search regions.

To assess the performance of FastCNP, we carry out an experimental study on 16 benchmark instances. Experimental results show that, despite its simplicity, FastCNP achieves a competitive performance compared to the state-of-the-art results. In particular, FastCNP finds new upper bounds for 6 instances and matches the best-known results for 9 out of the remaining 10 instances.

2 THE PROPOSED ALGORITHM

The proposed FastCNP algorithm follows the general iterated local search scheme and consists of three key components (Algorithm 1) - an initialization procedure, a fast neighborhood search procedure and a destructive-constructive procedure. FastCNP starts from a random initial solution. Then at each iteration, FastCNP first employs the fast neighborhood search procedure to attain a local optimal solution. Then it invokes the destructive-constructive procedure which attempts to move away from the current search region and jump to a new search region. The above steps are repeated until a stopping condition (e.g., a time limit t_{max}) is met.

Algorithm 1 A fast heuristic algorithm for CNP

```
1: Input: graph $G = (V, E)$, integer $K$, maximum allowed time limit $t_{max}$
2: Output: the best solution $S^*$ found
3: $S \leftarrow$ initialization() //generate an initial solution
4: $S^* \leftarrow S$
5: while $t_{max}$ is not reached do
6:     $S \leftarrow$ fast_neighborhood_search($S$) //local optimization
7:     if $f(S) < f(S^*)$ then
8:         $S^* \leftarrow S$
9:     end if
10:    $S \leftarrow$ destructive-constructive($S$) //perturb the solution
11: end while
```

*Corresponding author, also affiliated with the Institut Universitaire de France.

To efficiently explore the search space, the FastCNP algorithm relies on the fast (low-complexity) two-phase node exchange strategy described below. Contrary to the conventional node exchange (or swap) which results in a neighborhood of size $K(|V| - K)$, our two-phase node exchange leads to a smaller neighborhood and thus much fast. Specifically, the two-phase exchange strategy breaks the exchange of a pair of nodes into two phases. Let S be the current solution (the set of critical nodes), the "add" phase randomly selects one *large* connected component, and then adds to S a random node from the selected component regardless of its influence on the objective function. The "removal" phase moves a node from S to the residual graph such that this causes the minimum increase in the objective function. Using this two-phase exchange strategy, the effort required to examine the candidate solutions is greatly decreased. Our neighborhood search technique shares ideas with the neighborhood decomposition strategy used in [7].

The destructive-constructive procedure of FastCNP generates a new starting solution for the next round of neighborhood search by modifying slightly the input local optimal solution. Specifically, we randomly remove p_ω nodes (p_ω is a parameter called *perturbation strength*) from S and add them back to the residual graph; then we expand S with nodes randomly taken from a *large* component until the cardinality of S reaches K.

3 COMPUTATIONAL RESULTS

We assess the performance of FastCNP on the synthetic benchmark [1] largely tested in the literature [1, 2, 4]. FastCNP was implemented in C++, and complied using GNU gcc 4.1.2 with '-O3' option on a 2.5 GHz Intel E5-2670 processor with 2 GB RAM under Linux. Due to the stochastic nature of FastCNP, each instance was solved 30 times independently. For each run, we used a time limit $t_{max} = 3,600$ seconds (this condition is comparable to the stopping conditions used by the reference algorithms [1, 2, 4]).

To report our results, we indicate the best objective value among the 30 results ("f_{best}"), average objective value ("f_{avg}"), average time to find the best value for the 30 trials ("t_{avg}") and number of successful trials to find the best value ("*#succ*") of the 30 trials. Table 1 summarizes the computational results of FastCNP where *BKV* indicates the current best-known values (upper bounds) reported in the literature. In this table, the best values are highlighted in bold and the improved best upper bounds are marked by "⋆".

As we can see from Table 1, FastCNP obtains the best-known objective values for all instances except for WS1000 and FF2000 (for FF2000 whose proved optimum is 4545, the result of FastCNP is only one unit higher). In particular, FastCNP discovers 6 new best upper bounds (marked by "⋆" in Table 1). It is interesting to observe that the average objective values of these 6 improved results are also better than the previous best-known objective values. The average times to find the best objective values range from 0 to about 2,100 seconds, which are very competitive compared to the time consumed by the reference algorithms [1, 2].

Table 1: Performance of FastCNP with $t_{max} = 3,600$ seconds.

Instance	K	BKV	f_{best}	f_{avg}	t_{avg}	#succ
BA500	50	**195***	**195**	195.0	0.0	30/30
BA1000	75	**558***	**558**	558.0	43.4	30/30
BA2500	100	**3704***	**3704**	3713.0	308.7	15/30
BA5000	150	**10196***	**10196**	10202.9	18.1	22/30
ER235	50	**295***	**295**	295.0	11.6	30/30
ER466	80	1542	**1524**⋆	1524.0	378.9	30/30
ER941	140	5120	**5012**⋆	5014.7	1405.8	22/30
ER2344	200	997839	**959500**⋆	978643.7	1796.4	1/30
FF250	50	**194***	**194**	194.0	3.3	30/30
FF500	110	**257***	**257**	258.4	123.3	7/30
FF1000	150	**1260***	**1260**	1261.3	26.2	25/30
FF2000	200	**4545***	4546	4557.6	1088.5	2/30
WS250	70	3240	**3101**⋆	3192.5	1457.1	1/30
WS500	125	2130	**2078**⋆	2085.5	1673.7	1/30
WS1000	200	**113638**	120411	127675.1	2051.7	1/30
WS1500	265	13662	**13167**⋆	13267.7	2099.7	1/30

* Optimal results obtained within 5 days [1].
⋆ Improved best upper bounds.

4 CONCLUSIONS

We presented a fast heuristic algorithm for the computationally challenging critical node detection problem. The proposed FastCNP algorithm is based on an original component-based two-phase node exchange strategy which ensures an effective local optimization. Compared to the conventional two-node exchange (swap) strategy, the two-phase node exchange strategy is of much lower computational complexity, which contributes greatly to the search efficiency of the neighborhood search procedure. The computational results on 16 popular benchmark instances showed that FastCNP is highly competitive compared with state-of-the-art results and in particular is able to find 6 improved best solutions (new upper bounds). Finally, given the reported results for CNP, it would be interesting to investigate the usefulness of the component-based two-phase node exchange strategy introduced in this work in the context of other critical node problems.

[1]It is available at: http://individual.utoronto.ca/mventresca/cnd.html

ACKNOWLEDGMENTS

The authors would like to thank the referees for their useful comments and suggestions. Support for Yangming Zhou from the China Scholarship Council (2014-2018) is also acknowledged.

REFERENCES

[1] Roberto Aringhieri, Andrea Grosso, Pierre Hosteins, and Rosario Scatamacchia. 2016. A general evolutionary framework for different classes of Critical Node Problems. *Engineering Applications of Artificial Intelligence* 55 (2016), 128–145.
[2] Roberto Aringhieri, Andrea Grosso, Pierre Hosteins, and Rosario Scatamacchia. 2016. Local search metaheuristics for the critical node problem. *Networks* 67, 3 (2016), 209–221.
[3] Ashwin Arulselvan, Clayton W. Commander, Lily Elefteriadou, and Panos M. Pardalos. 2009. Detecting critical nodes in sparse graphs. *Computers & Operations Research* 36, 7 (2009), 2193–2200.
[4] Wayne Pullan. 2015. Heuristic identification of critical nodes in sparse real-world graphs. *Journal of Heuristics* 21, 5 (2015), 577–598.
[5] Marco Di Summa, Andrea Grosso, and Marco Locatelli. 2011. Complexity of the critical node problem over trees. *Computers & Operations Research* 38, 12 (2011), 1766–1774.
[6] Marco Di Summa, Andrea Grosso, and Marco Locatelli. 2012. Branch and cut algorithms for detecting critical nodes in undirected graphs. *Computational Optimization and Applications* 53, 3 (2012), 649–680.
[7] Yangming Zhou, Jin-Kao Hao, and Béatrice Duval. 2017. Opposition-based memetic search for the maximum diversity problem. *IEEE Transactions on Evolutionary Computation* PP, 99 (2017), 1–1.