VNS solutions for the Critical Node Problem

Roberto Aringhieri, Andrea Grosso, Pierre Hosteins [1,2]

Dipartimento di Informatica
Università degli Studi di Torino
Turin, Italy

Rosario Scatamacchia [1,3]

Dipartimento di Automatica e Informatica
Politecnico di Torino
Turin, Italy

Abstract

We present a VNS algorithm for the Critical Node Problem, i.e., the maximal fragmentation of a graph through the deletion of k nodes. Two computational efficient neighbourhoods are proposed proving also their equivalence to the straightforward exchange of two nodes. The results of the proposed VNS algorithms outperform those currently available in literature.

Keywords: Critical Node Problem, graph fragmentation.

[1] Work supported by a Google Focused Grant on Mathematical Programming, project "Exact and Heuristic Algorithms for Detecting Critical Nodes in Graphs".
[2] Email: `roberto.aringhieri@unito.it`, `grosso@di.unito.it`, `hosteins@di.unito.it`
[3] Email: `rosario.scatamacchia@polito.it`

1 Introduction

Given an undirected graph $G(V, E)$ and an integer k, the Critical Node Problem (**CNP**) consists in finding a subset of k nodes $S \subseteq V$, such that the number of node pairs still connected in the induced subgraph $G[V \setminus S]$

$$f(S) = |\{u, v \in V \setminus S \colon u \text{ and } v \text{ are connected by a path in } G[V \setminus S]\}|$$

is as small as possible.

To the authors' knowledge, the problem can be traced back to the so-called *network interdiction problems* studied by Wollmer [14] and later Wood [15] where anyway emphasis is placed on arc deletion instead of nodes. The interest about node deletions seems to have spread more recently: Borgatti [3] states the CNP in the context of detecting so-called "key players" in a relational network; Arulselvan et al. [1] and Ventresca [13] emphasize contagion control via vaccination of a limited number of individuals, when the nodes of the graph are potentially infected individuals and the edges represent contacts occurring between them. Similar problems are studied for assessing robustness of communication networks by Dinh et al. [5,11].

The CNP is known to be NP-complete [1], polynomially solvable on trees [4] and other "simple" kinds of graphs [2,10]. For a broad literature review, including problems with different metrics about graph fragmentation, we refer to comprehensive references given in other works, for example [10,9]. From a practical point of view, the CNP on general graphs has been tackled by Arulselvan et al. [1] by proposing a MILP model and a heuristic approach based on a greedy heuristic coupled with a successive local search phase. Sophisticated metaheuristics – namely population-based incremental learning and simulated annealing – are studied and experimentally compared by Ventresca [13].

In this paper, we present a solution framework for the CNP based on the Variable Neighbourhood Search (**VNS**) methodology [7,8]. The main contribution of this paper concerns the development of two computational efficient neighbourhoods which are also proved to be equivalent to the straightforward neighbourhood exchanging of two nodes u and v in such a way that $u \in S$ and $v \in V \setminus S$. This allows the development of a VNS solution framework for the CNP which largely improves the best known solutions provided in literature. The paper is organized as follows. Section 2 reports the two efficient neighbourhood explorations. Section 3 depicts the proposed VNS approach for the solution of CNP. Section 4 discusses the computational results of the proposed algorithm on the state of the art benchmark instances. Finally, Section 5 closes the paper.

2 Efficient neighbourhood exploration

A straightforward neighbourhood, say N_0, tries to improve a given solution S by exchanging a node $u \in S$ with another node $v \in V \setminus S$ (*2-node-exchange*). Its main drawback lies in the computational complexity of the move evaluation which requires to perform a graph search whose complexity is $\mathcal{O}(|V| + |E|)$. In the worst case, that is when $k = |S| = \frac{|V|}{2}$, the whole neighbourhood exploration is $\mathcal{O}(|V|^3)$ because of the number of exchanges to be evaluated is quadratic in the number of nodes in V. We propose two more efficient neighbourhoods based on the idea of reducing the number of exchanges to be evaluated to overcome this computational issue.

The former, say N_1, considers all nodes in $u \in S$ and evaluates, for each node, the exchange with the node $v \in V \setminus S$ that disconnects the graph as much as possible, i.e., such that $v = \arg\max\{f((S \cup \{v'\}) \setminus \{u\}) - f(S)\}$. The latter, say N_2, builds a move exchanging each node $v \in V \setminus S$ with the node u which yields the minimum increase in the objective function, i.e., $u = \arg\min\{f((S \cup \{v\}) \setminus \{u'\}) - f(S)\}$. Note that, both N_1 and N_2 break ties at random.

The number of exchanges in N_1 and N_2 are, respectively, k and $|V| - k$. Therefore, the complexity of a complete neighbourhood exploration is $\mathcal{O}(|V|^2)$ instead of $\mathcal{O}(|V|^3)$ of 2-node-exchange. This is due to the fact that v' and u' can be computed in $\mathcal{O}(|V| + |E|)$ through a modified depth-first search [12].

Finally, we would like to remark that N_0, N_1 and N_2 yield the same best move (u^*, v^*), unless breaking ties lead to the choice of different pairs of nodes, as proved below.

Theorem 2.1 *Neighbourhood exploration methods N_0, N_1 and N_2 all yield the same result (u^*, v^*) when there is only one best move (no ties).*

Proof. Let us suppose there exists one and only one best possible move (u^*, v^*). By construction N_0 is sure to identify it. Let us now consider N_1: if we select a node $u \neq u^*$ from set S, any resulting move (u, v) the algorithm finds is by construction inferior to (u^*, v^*). If in fact we choose $u = u^*$, let us suppose the algorithm extracts a pair (u^*, v) where $v \neq v^*$: since the algorithm evaluates the impact of all possible nodes $v \in (V \setminus S) \cup \{u\}$, it means we have found a pair (u^*, v) with higher impact than (u^*, v^*), which is impossible by construction.
Thus N_1 extracts the best possible pair (u^*, v^*). Given that for a given $v \in V \setminus S$, the rule based on N_2 evaluates the impact of all nodes $u \in S \cup \{v\}$, a similar reasoning proves that N_2 will also select (u^*, v^*). □

3 A Variable Neighbourhood Search for CNP

In this section we present a general VNS solution framework to deal with CNP. In the following we refer to the basic VNS scheme discussed in [8] (cf., algorithm 7). From a notational point of view, we use h instead of k to denote the kth neighbourhood of a solution x, and S to denote a solution.

The main ingredients of our algorithm are: the procedure to compute an initial solution, the improvement and the shake procedures. The pseudocode of our algorithm is depicted in Algorithm 1.

The initial solution can be any set S of deleted k nodes. In our current implementation, we select such nodes by applying the greedy algorithm proposed in [1]. The improvement procedure is a Local Search (line 3) based on the first improvement exploration of the neighbourhood N_1 or, alternatively, N_2. Let ϕ_u be the occurrence or frequency in which u belongs to a solution S. The value ϕ_u is updated ($\phi_u \longleftarrow \phi_u + 1$) in two cases: if u is added to the solution S during the neighbourhood exploration (line 3), and if u belongs to a solution improving the current best solution (line 5). The shaking procedure replaces the h most frequent nodes in S with the h least frequent nodes in $V \setminus S$.

Algorithm 1 *A Variable Neighbourhood Search for CNP*

```
**CNP-VNS** ($S$, $t_{\max}$, $h_{max}$)
repeat
1  $h \longleftarrow 2$;
   repeat
2     $S' \longleftarrow$ Shake($S$, $h$);
3     $S'' \longleftarrow$ FirstImprovement($S'$, $N_1$);
4     if ($f(S'') < f(S)$) then
5        $S \longleftarrow S''$; $h \longleftarrow 2$;
      else
6        $h \longleftarrow h + 1$;
      end;
   until $h = h_{\max}$;
7  $t \longleftarrow$ cpuTime();
until $t \geq t_{\max}$;
return $S$
```

In the following, we consider 4 different versions derived from Algorithm 1 varying the neighbourhood in the Local Search (N_1 or N_2) and increasing h

($h = 2, \dots, h_{\max}$) or decreasing h ($h = h_{\max}, \dots, 2$). We denote them as VNS-I-N_1, VNS-I-N_2, VNS-D-N_1, VNS-D-N_2 where I and D stand for increasing and decreasing h, respectively.

4 Computational analysis

In this section we report the computational analysis of the 4 versions of Algorithm 1, with two different update mechanisms of the frequencies of the selected nodes. They were programmed in standard C++ and compiled with `gcc 4.1.2`. All tests were performed on an HP ProLiant DL585 G6 server with two 2.1 GHz AMD Opteron 8425HE processors and 16 GB of RAM. We use the graphs presented in [13] as benchmark instances and compare our results with the best known results (coming from [6]), provided in our tables in the column "BK". Each graph has a specific topology based on Erdos-Renyi, Barbasi-Albert, Watts-Strogatz and Forest Fire models (see [13] for more details). In the column "graph" we indicate the type of graph by two letters (e.g. BA stands for Barabasi-Albert, etc...), followed by its number of nodes. New best known results are displayed in bold font.

The results are clearly in favour of the VNS algorithms, with a few very remarkable improvements. The gap between the VNS and the best known result is often around 50% and sometimes much higher, except for BA graphs where it can be only a few percent lower.

In Table 2, we report the results of our algorithms when adopting a different policy – less pervasive – for the updating of frequencies: actually, we update ϕ_u only when u belongs to a solution improving the current best solution (line 5 of Algorithm 1). Thus, while we previously updated the frequency of a node each time it was found in a solution, we now only update it when it is found in a local optimum. Comparing the results with those in Table 1, it is evident that there is not a clear dominance of the updated method, in the sense that the conclusion will differ according to the family of graphs considered. We also observe that the number of best known solutions are almost the same, that is 22 in Table 1 and 24 Table 2.

Running times can be quite long when the graphs reach 1000 nodes so we choose to terminate the algorithm after 10000 seconds. Even though this seems substantial, they are comparable or inferior to meta-heuristics from [13], however the newer algorithms in [6] usually run in a matter of seconds as greedy algorithms usually do. Note that VNS based on N_1 are faster (in terms of running time) than those based on N_2 when $k < \frac{|V|}{2}$. This is true for all our test instances.

A version with best improvement instead of first improvement gives similarly good results (although it finds a lower number of best known results), however the running times are higher for almost all instances: on average from 45% to 244% depending on the algorithm.
Table 3 summarizes the best known solution computed by the 4 versions of the algorithm both in the case of first and best improvements (results from the best improvement versions are displayed in italic).

5 Conclusions

We devised a solution framework based on VNS methodology for the Critical Node Problem. Our solution framework exploits the two efficient neighbourhoods discussed in Section 2. The results proposed in Table 1 and Table 2 largely outperform those reported in literature, especially for harder benchmark instances. Running times can be high but are justified by the improve-

graph	K	BK	VNS-D-N_1	VNS-D-N_2	VNS-I-N_1	VNS-I-N_2
BA500	50	203	**195**	**195**	**195**	**195**
BA1000	75	580	**559**	**559**	**559**	**559**
BA2500	100	4254	3722	**3704**	3722	**3704**
BA5000	150	11886	**10196**	10218	**10196**	**10196**
ER235	50	1141	306	306	301	**298**
ER466	80	19952	1562	1611	1561	1725
ER941	140	114166	5470	6106	8106	**5198**
ER2344	200	1606656	1112994	1091185	1118785	1094239
WS250	70	13786	7175	11196	10237	12457
WS500	125	53779	**2148**	2209	2230	2209
WS1000	200	308596	198494	**139653**	268500	179531
WS1500	265	653015	16210	16549	14623	**14619**
FF250	50	302	**194**	198	198	198
FF500	110	344	**257**	258	**257**	**257**
FF1000	150	1880	1270	1274	**1263**	1265
FF2000	200	7432	4578	4584	4583	**4549**

Table 1
Results of the 4 algorithms (BK stands for Best Known result from previous works, while K is the number of nodes deleted from the graph).

graph	K	BK	VNS-D-N_1	VNS-D-N_2	VNS-I-N_1	VNS-I-N_2
BA500	50	203	**195**	**195**	**195**	**195**
BA1000	75	580	**559**	**559**	**559**	**559**
BA2500	100	4254	**3722**	**3704**	**3722**	**3704**
BA5000	150	11886	**10196**	10218	**10196**	10218
ER235	50	1141	303	335	301	302
ER466	80	19952	**1542**	1727	1567	1751
ER941	140	114166	5503	6289	5658	5628
ER2344	200	1606656	1067397	1097573	1052406	**1034333**
WS250	70	13786	8833	**6610**	10413	7186
WS500	125	53779	2170	2199	2152	2213
WS1000	200	308596	200225	256239	255061	154813
WS1500	265	653015	17198	26225	14719	15692
FF250	50	302	**194**	199	**194**	199
FF500	110	344	**257**	258	**257**	258
FF1000	150	1880	1270	1274	1270	1273
FF2000	200	7432	4576	4584	4577	4550

Table 2
Results of the 4 algorithms (update ϕ_u only on line 5 of Algorithm 1).

graph	BA500	BA1000	BA2500	BA5000	ER235	ER466	ER941	ER2344
BK	195	559	3704	*10122*	*297*	1542	5198	1034333
graph	FF250	FF500	FF1000	FF2000	WS250	WS500	WS1000	WS1500
BK	194	257	*1260*	4549	6610	2148	139653	14619

Table 3
New Best Known

ment of the solution quality.

Ongoing work is devoted to devise auxiliary data structure or different neighbourhoods, which could improve the average case complexity, possibly trading a loss in the quality of the solution for a reduction of the computational time.

References

[1] A. Arulselvan et al, *Detecting critical nodes in sparse graphs*, Computers & Operations Research **36** (2009), pp. 2193–2200.

[2] B. Addis, M. D. S. and A. Grosso, *Removing critical nodes from a graph: complexity results and polynomial algorithms for the case of bounded treewidth*, Discrete Applied Mathematics **16-17** (2013), pp. 2349–2360.

[3] Borgatti, S. P., *Identifying sets of key players in a network*, Computational and Mathematical Organization Theory **12** (2006), pp. 21–34.

[4] Di Summa, M., A. Grosso and M. Locatelli, *The critical node problem over trees*, Computers and Operations Research **38** (2011), pp. 1766–1774.

[5] Dinh, T. N. and M. T. Thai, *Precise structural vulnerability assessment via mathematical programming*, in: *MILITARY COMMUNICATIONS CONFERENCE, 2011-MILCOM 2011*, IEEE, 2011, pp. 1351–1356.

[6] Edalatmanesh, M., "Heuristics for the Critical Node Detection Problem in Large Complex Networks," Ph.D. thesis, Faculty of Mathematics and Science, Brock University, St. Catharines, Ontario (2013).

[7] Hansen, P., N. Mladenović and J. A. M. Pérez, *Variable neighbourhood search: methods and applications*, 4OR **6** (2008), pp. 319–360.

[8] Hansen, P., N. Mladenović and J. A. M. Pérez, *Variable neighbourhood search: methods and applications*, Ann Oper Res **175** (2010), pp. 367–407.

[9] S. Shen et al, *Exact interdiction models and algorithms for disconnecting networks via node deletions*, Discrete Optimization **9** (2012), pp. 172–88.

[10] Shen, S. and J. Cole Smith, *Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs*, Networks **60** (2012), pp. 103–119.

[11] T. N. Dinh et al, *On new approaches of assessing network vulnerability: Hardness and approximation on approximation of new optimization methods for assessing network vulnerability*, IEEE/ACM Transactions on Networking **20** (2012), pp. 609–619.

[12] Tarjan, J., *Efficient algorithms for graph manipulation*, Communications of the ACM **16** (1973), pp. 372–378.

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

[14] Wollmer, R., *Removing arcs from a network*, Operations Research **12** (1964), pp. 934–940.

[15] Wood, R. K., *Deterministic network interdiction*, Mathematical and Computer Modelling **17** (1993), pp. 1–18.