J Heuristics
DOI 10.1007/s10732-015-9290-5

Heuristic identification of critical nodes in sparse real-world graphs

Wayne Pullan[1]

Received: 19 April 2014 / Revised: 21 March 2015 / Accepted: 14 April 2015

Abstract Given a graph, the critical node detection problem can be broadly defined as identifying the minimum subset of nodes such that, if these nodes were removed, some metric of graph connectivity is minimised. In this paper, two variants of the critical node detection problem are addressed. Firstly, the basic critical node detection problem where, given the maximum number of nodes that can be removed, the objective is to minimise the total number of connected nodes in the graph. Secondly, the cardinality constrained critical node detection problem where, given the maximum allowed connected graph component size, the objective is to minimise the number of nodes required to be removed to achieve this. Extensive computational experiments, using a range of sparse real-world graphs, and a comparison with previous exact results demonstrate the effectiveness of the proposed algorithms.

Keywords Real world graph · Critical node detection · Heuristic algorithm

1 Introduction

Critical node detection problems (CNPs) are NP-complete for general graphs (Arulselvan et al. 2009; Ventresca 2012; Ventresca and Aleman 2014; Arulselvan et al. 2011; Fan and Pardalos 2010; Ventresca and Aleman 2014) and require the optimal deletion of nodes from a graph such that some metric of graph connectivity is minimised. Applications related to critical node detection problems include epidemic control (Medlock and Galvani 2009) where the graph represents the contacts between individuals or

✉ Wayne Pullan
w.pullan@griffith.edu.au

[1] School of Information and Communication Technology, Griffith University, Gold Coast, QLD, Australia

Published online: 26 April 2015

groups and the optimal selection of nodes to be immunised are those that minimise the number of paths between nodes that do not pass through an immunised node. Other applications include identifying critical nodes in communication networks (Sun and Shayman 2007; Nguyen et al. 2013; Arulselvan et al. 2009; Dinh et al. 2014; Shen et al. 2012), critical persons in social or terrorist networks (Krebs 2002; Medlock and Galvani 2009; Nguyen et al. 2013), and critical proteins in protein–protein interaction networks (Boginski and Commander 2008). Typically, the critical node detection problem will have applications in any diffusive process that can be modelled using a graph.

The two variants of the critical node detection problem addressed in this study are now defined relative to the undirected graph $G(V, E)$ where $V = \{1, \ldots, n\}$ is the set of n nodes and $E \subseteq V \times V$ is the set of edges in G:

- CNP1: Given an integer K, identify a subset $S \subseteq V$ where $|S| \leq K$, such that the total number of connected node pairs in the induced subgraph $G[V \backslash S]$ is minimised. The metric to be minimised for CNP1 is the percentage $C_1 = 100 \sum_{i=1}^{T} \binom{|P_i|}{2} / \binom{|V|}{2}$ where T is the total number of connected components (P_i) in the induced subgraph $G[V \backslash S]$.
- CNP2: Given an integer L, identify the minimal subset $S \subseteq V$ such that the largest connected component in the induced subgraph $G[V \backslash S]$ contains no more than L nodes (i.e. $|P_i| \leq L, 1 \leq i \leq T$) where T is the total number of connected components (P_i) in the induced subgraph $G[V \backslash S]$. The metric to be minimised for CNP2 is $C_2 = |S|$, the number of nodes removed from G. This variant is referred to as the cardinality constrained critical node detection problem and was first proposed in Arulselvan et al. (2011)

Linear 0–1 versions of these two problems have been formulated (Veremyev et al. 2014) using the following definitions:

> For each pair of nodes $i, j \in V$, let $u_{ij} = 1$ if nodes i and j are connected by a path (0 otherwise) and, for any node $i \in V$, let $v_i = 1$ if node $i \in S$ (0 otherwise).

CNP1 can be formulated as:

$$\text{minimise } C_1 \tag{1}$$

$$\text{subject to: } u_{ij} + v_i + v_j \geq 1, \quad \forall (i, j) \in E \tag{2}$$

$$u_{ik} + u_{kj} - u_{ij} \leq 1, \quad \forall i, j, k \in V \tag{3}$$

$$\sum_{i \in V} v_i \leq K, \tag{4}$$

$$v_i, u_{ij} \in \{0, 1\} \quad \forall i, j \in V \tag{5}$$

and CNP2 can be formulated as:

$$\text{minimise } C_2 \tag{6}$$

$$\text{subject to: } u_{ij} + v_i + v_j \geq 1, \quad \forall (i, j) \in E \tag{7}$$

$$u_{ik} + u_{kj} - u_{ij} \leq 1, \qquad \forall i, j, k \in V \tag{8}$$

$$\sum_{j \in V} u_{ij} \leq L - 1, \qquad \forall i \in V \tag{9}$$

$$v_i, u_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{10}$$

Relatively few exact and heuristic methods have been proposed for the critical node detection problem (Borgatti 2006; Dinh et al. 2010, 2012; Summa et al. 2011, 2012; Shen et al. 2013; Addis et al. 2013) and most have addressed small or special graphs (e.g. trees). A detailed computational study is presented in Arulselvan et al. (2009) which includes both an integer programming model and a heuristic that starts with $S = V \backslash M$, where M is a maximal independent set of V. Nodes are then greedily selected and removed from S until $|S| \leq K$. A local search is then performed using S as the starting point. The largest graphs in this study contained 150 nodes and the heuristic produced high quality solutions in a fraction of the time required by the commercial integer programming solver CPLEX.

More recently, in Ventresca (2012), simulated annealing and a population based incremental learning algorithm were applied to the critical node problem with graphs of up to 5000 nodes. However, as noted in Ventresca (2012), "comparison of our results to pre-existing approaches was not possible due to either limited RAM or exceptionally high computational demands". In addition, graph generators were used to generate the graphs studied which makes exact replication of the test graph benchmarks difficult.

In Veremyev et al. (2014) it was observed that all available exact methods for solving critical node problems for the general case of an undirected graph deal with the linear 0–1 formulations listed in Eqs. 1–10 included the $\mathcal{O}(n^3)$ triangular connectivity constraints (Eqs. 3 and 8). This has restricted exact solution methods based on these linear 0–1 formulations to optimally solving CNP1 and CNP2 problems for small sparse graphs of up to 150 nodes (problems of this size are easily solved by heuristic methods). As a means of overcoming this limitation, a compact linear 0–1 formulation was developed that replaced the $\mathcal{O}(n^3)$ constraint with a $\mathcal{O}(n^2)$ constraint. In addition, reformulations and valid inequalities were identified that improved the performance of their exact solution methods, designated as CNP1c-M for problem CNP1 and CNP2c-M for problem CNP2. These two solution methods were able to find exact solutions for real-world sparse networks up to 10 times larger and with CPU time up to 1000 times faster when compared to previous studies. All graphs used in Veremyev et al. (2014) were obtained from the University of Florida Sparse Matrix Collection (Davis and Hu 2011) and their exact method was able to solve the CNP1/CNP2 problems for graphs with up to 1138 nodes. It should be noted that graphs of this size are still difficult for state of the art heuristic CNP algorithms to solve.

Given that exact solutions are now available for larger, readily available real-world graphs, an effective benchmark now exists for the evaluation of heuristic critical node detection algorithms. Utilising this benchmark, a major goal of this study was to produce the following two CNP benchmarks for the evaluation of CNP heuristic algorithms:

Benchmark 1: This benchmark consists of all the graphs from Veremyev et al. (2014) where the exact solutions were provided with a precision of either ± 0.05 or ± 0.005. To create a more difficult benchmark, all CNP1 results in this paper have a precision of ± 0.00005 which, when rounded to either ± 0.05 or ± 0.005 precision, with two exceptions, match those presented in Veremyev et al. (2014). This extension of precision provides a much more difficult benchmark for future heuristics as it is relatively straight-forward for a heuristic to attain the ± 0.05 and ± 0.005 (Veremyev et al. 2014) precision results but much more difficult to attain the CNP1 ± 0.00005 results.

Benchmark 2: This benchmark consists of larger graphs than those of Benchmark 1 with all graphs obtained from the University of Florida Sparse Matrix Collection (Davis and Hu 2011). As there are no exact CNP results available for these graphs, long-running experiments with the algorithms of this paper were first used to identify the best solutions attainable and then these algorithms were evaluated by identifying the best possible solutions that CNA1 and CNA2 could attain over 100 successful consecutive trials. The performance of these algorithms on Benchmark 1 does give some grounds for confidence that the CNA1/CNA2 solutions for Benchmark 2 are of reasonable quality and are suitable for the evaluation of future CNP algorithms.

The structure of these two benchmarks is in-line with the observation that the history of solving problems such as the CNP typically goes through a number of evolutions. The first evolution is smaller problems that can be solved by exact methods and which are normally trivial for most heuristics methods. The second evolution is larger problems that are beyond the reach of exact methods but with some heuristic achieving new benchmark solutions which then leads to the third, repetitive evolution where new heuristics attempt to achieve these results using less CPU time (with the occasional discovery of new results and larger problems being introduced). However, after the first evolution, the quality of the achieved results is generally unknown as exact methods are unable to solve these larger problems.

Another factor which these benchmarks address is that, in the past, most CNP heuristics have been evaluated using generated model graphs (e.g. Forest Fire). The issue here is that, given the number of parameters that need to be supplied to generate these graphs, the possibility of different versions of the graph generator and the different computer systems used, there is no guarantee that identical graphs have been generated when performing a comparison of heuristics. For the CNP, small differences in graphs can significantly affect the difficulty of locating a solution. However, by using the actual graphs, directly downloaded from the University of Florida Sparse Matrix Collection (Davis and Hu 2011), there is no doubt that graphs will be identical for different studies.

The remainder of this paper is organised as follows: in Sect. 2, two critical node algorithms that address problems CNP1 and CNP2 are presented and discussed while Sect. 3 contains experimental results for these algorithms which are compared with the exact results attained in Veremyev et al. (2014) (Benchmark 1) and new results also presented for larger graphs (Benchmark 2). Finally, Sect. 4 is a conclusion that summarises this paper and discusses opportunities for future research.

2 Algorithms

This section describes the two algorithms (CNA1/CNA2) presented in this paper where algorithm CNA1 addresses the CNP1 problem while algorithm CNA2 addresses the CNP2 problem.

Algorithm CNA1 performs a complete search when $K < 5$ by evaluating the removal of all possible combinations of nodes (lines 3–5). When $K \geq 5$, algorithm CNA1 is a multi-start greedy algorithm for solving the CNP1 problem. This algorithm focuses on the larger graph components and randomly removes nodes in $G[V \backslash S]$ (lines 7–9) and adds back nodes that cause the least increase in C_1 (lines 10–12). To prevent search stagnation, periodically a partial restart is performed (lines 16–18). C_1 is continually updated within CNA1 whenever a node is added or removed from S.

Algorithm 1: CNA1 : Addresses the CNP1 problem defined in Sect. 1.

Input: An integer K specifying the maximum number of nodes which may be removed from the graph, the target objective value C_1^T, and an undirected graph $G(V, E)$.
Output: The minimum C_1 value achieved.
Data: $C_1 = 100 \sum_{i=1}^{T} \binom{|P_i|}{2} / \binom{|V|}{2}$ is the current objective value which is always updated when S changes, T is the number of graph components, P_i is the ith graph component, and S is the set of nodes removed from G.

```
1  S ← ∅
2  Step ← 0
3  if |K| < 5 then
4      C1 ← CompleteSearch
5      return C1
6  while C1 > C1^T do
7      C1^S ← C1
8      P ← SelectComponent (G[V\S])
9      v ← SelectRemoveNode (P)
10     S ← S ∪ {v}
11     if |S| > K then
12         v ←SelectAddNode
13         S ← S\{v}
14     if C1 < C1^S then
15         Step ← 0
16     Step ← Step + 1
17     if C1 > C1^T & Step > 50K then
18         AddNodes
19         Step ← 0
20 return C1
```

The functions used in Algorithm CNA1 are:

- *CompleteSearch*: Invoked when $K < 5$ and performs a recursive complete search by evaluating the effect of removing all possible combinations of K nodes.
- *SelectComponent*: Performs a random selection of a graph component from a list of the larger components in $G \backslash S$. In this context, the larger components are

defined as those whose size is greater than $N_{max} - (N_{max} - N_{min})/2$ where N_{max} is the largest component size and N_{min} is the smallest component size.
- *SelectRemoveNode*: Uniform randomly selects a node from the component chosen by *SelectComponent*.
- *SelectAddNode*: Invoked when $|S| > K$ or a partial restart is performed and identifies the node whose removal from S gives the minimum increase in C_1. The evaluation of the increase in C_1 for each node in S is performed by scanning the adjacency list of the node to determine if adding the node will cause component recombinations (as the node being added must be in the same component as all its neighbours (except those that are also in S)). This operation is $\mathcal{O}(|S|L_i)$ where $|S| \leq K + 1$ and L_i is the length of the adjacency list for node i which, for sparse graphs, is typically small.
- *AddNodes*: Invoked if there has been no improvement in C_1 for the last $50K$ node selections. A partial restart is performed by removing 25 % of nodes from S. The nodes to be removed are selected using *SelectAddNode*.

The motivation for the stopping criteria in CNA1 was that it could be used to achieve the results obtained by the exact method of Veremyev et al. (2014). When CNA1 was used to produce new target C_1^T values for Benchmark 2, the stopping criteria became, in effect, no improvement over an extended period of time.

Algorithm CNA2 is a multi-start greedy algorithm for solving the CNP2 problem. This algorithm has two phases: the initial phase repeatedly removes nodes from G until all graph components have L or fewer nodes (lines 2–5). Using $K = |S| - 1$ as the target value for the number of nodes removed (line 9), the second phase removes (lines 13–15) and adds (lines 16–18) nodes until the number of nodes removed equals K and all graph components have L or fewer nodes. This process is then repeated until K equals the target value C_2^T. To prevent search stagnation, periodically a partial restart is performed (lines 22–24). $F = \sum_{i=1}^{T}(P_i - L)u_i$, the total number of nodes in excess of L in all graph components is continually updated within CNA2 whenever a node is added to or removed from S. The functions used in Algorithm CNA2 are:

- *SelectComponent*: Performs a random selection of a graph component from a list of the larger components in $G \backslash S$. In this context, the larger components are defined as those components whose size is within two of the size of the largest component.
- *SelectRemoveNode*: Uniform randomly selects a node from the component chosen by *SelectComponent*.
- *SelectAddNode*: Invoked when $|S| > K$ or a partial restart is performed and identifies the node whose removal from S gives the minimum increase in F. The evaluation of the increase in F for each node in S is performed by scanning the adjacency list of the node to determine if adding the node will cause component recombinations (as the node being added must be in the same component as all its neighbours (except those that are also in S)). This operation is $\mathcal{O}(|S|L_i)$ where $|S| \leq K + 1$ and L_i is the length of the adjacency list for node i which, for sparse graphs, is typically small.

Algorithm 2: CNA2 : Addresses the CNP2 problem defined in Sect. 1.

Input: An integer L specifying the largest allowed connected graph component, the target objective value for C_2^T and an undirected graph $G(V, E)$.

Output: The minimum C_2 value achieved.

Data: $F = \sum_{i=1}^{T}(P_i - L)u_i$ is the total number of nodes in excess of L in all graph components, T is the number of graph components, P_i is the ith graph component, let $u_i = 1$ if $|P_i| > L$ (0 otherwise), S is the set of nodes removed from G and F^S is the value of $\sum_{i=1}^{T}(P_i - L)u_i$ at the start of each iteration.

```
1  S ← ∅
2  while F > 0 do
3      P ← SelectComponent (G[V\S])
4      v ← SelectRemoveNode (P)
5      S ← S ∪ {v}
6  K ← |S|
7  Step ← 0
8  while K > C_2^T do
9      F^S ← F
10     K ← |S| − 1
11     v ←SelectAddNode (S)
12     S ← S\{v}
13     while F > 0 do
14         P ← SelectComponent (G[V\S])
15         v ← SelectRemoveNode (P)
16         S ← S ∪ {v}
17         if |S| > K then
18             v ←SelectAddNode (S)
19             S ← S\{v}
20         if F < F^S then
21             Step ← 0
22         Step ← Step + 1
23         if F > 0 & Step > 50K then
24             AddNodes (S)
25             Step ← 0
26 return K
```

– *AddNodes*: Invoked if there has been no improvement in F for the last $50K$ node selections and a partial restart is performed by randomly removing 25 % of nodes from S.

For greedy CNP heuristics, the most time consuming operation is the evaluation of the total effect of removing / adding nodes which normally involves a depth first search of the complete graph for each candidate node. However, unlike other greedy CNP heuristics, CNA1 and CNA2 operate at the graph component (P_i) or list of removed nodes (S) level where both $|P_i|$ and $|S|$ are typically much smaller than $|V|$ so this overhead is considerably reduced. The time for the actual evaluation of a node removal / addition is directly dependent on the degree of the node being removed / added and, for the sparse graphs evaluated in this study, node degree is almost always small. For example, using the β index ($= |E|/|V|$) as a measure of graph connectivity, the

largest β value in this study is 6.4 for the *USAir97* graph of 332 nodes. If this graph was complete then it would have $\beta = 165.5$.

For both CNA1 and CNA2, adding a node to S has the potential to cause graph component splits while removing a node from S has the potential to cause combinations of graph components. As both splitting and combining graph components are implemented using a depth-first search, their efficiency is directly proportional to the number of nodes and edges in the affected graph components.

As a note, the constant values used in CNA1/CNA2 were determined by running a number of experiments where the constant values were separately varied from small to medium to large. The actual values chosen were those that gave the best overall results for Benchmark 1.

3 Computational experiments

The major objectives of the computational experiments were to compare the results obtained by CNA1 and CNA2 with those presented in Veremyev et al. (2014) and to also extend these results to larger sparse real-world graphs as a benchmark for future algorithms. As in Veremyev et al. (2014), the sparse real-world graphs used in this paper were all sourced from the University of Florida Sparse Matrix Collection (Davis and Hu 2011) and are listed in the results tables along with $|V|$, $|E|$, and β for each graph.

In all tables in this study:

- K is the maximum number of nodes that can be removed from the graph for problem CNP1.
- C_1^M is the value attained by CNP1c-M (Veremyev et al. 2014) for CNP1.
- C_1^T is the target value for CNA1.
- L is the maximum graph component size allowed for problem CNP2.
- C_2^M is the value attained by CNP2c-M (Veremyev et al. 2014) for CNP2.
- C_2^T is the target value for CNA2.
- CPU is the processor time in seconds. For CNA1/CNA2, the processor time shown in all tables is the average over 100 consecutive successful trials each with different random initialisations.
- C_1^B is the best result obtained by CNA1 for problem CNP1.
- C_2^B is the best result obtained by CNA2 for problem CNP2.

As a basic guide for future comparisons with these results, all experiments were performed on a computer that, when executing the DIMACS Machine Benchmark[1] required 0.17 CPU seconds for r300.5, 1.08 CPU seconds for r400.5 and 4.12 CPU seconds for r500.5.

[1] *dmclique*, ftp://dimacs.rutgers.edu in directory /pub/dsj/clique

3.1 Preliminary results

With regard to small graphs, Table 1 shows comparative C_1 results attained by IP (Veremyev et al. 2014), the basic integer programming solution to CNP1 (Equations 1–5), CNP1c-M (Veremyev et al. 2014), the compact with additional constraints integer programming solution to CNP1 and CNA1 for three small graphs that have been studied in the past. For CNA1, for each graph and K value combination, 100 consecutive successful CNA1 trials were required for success where a trial of CNA1 was deemed successful when it found a C_1 that was less than or equal to $C_1^T + 0.00005$.

With regard to larger graphs, a recent study (Ventresca 2012) compared the results obtained by a Population Based Incremental Learning (PBIL) algorithm with results obtained by both Random Sampling and Simulated Annealing (SA). This study used 16 benchmark optimization instances based on Erdos–Renyi (ER), Barabasi–Albert (BA), Watts–Strogatz (WS) and Forest Fire (FF) algorithms (described in detail in Ventresca (2012)). This study was replicated in Edalatmanesh (2013) using an algorithm (DFSH-post) which is based on a depth-first search traversal of the graph with a specially designed ranking function that considers information local to each vertex. Table 2 compares the results attained by Random Sampling (Ventresca 2012), SA (Ventresca 2012), PBIL (Ventresca 2012), DFSH-post (Edalatmanesh 2013) with those attained by CNA1. For conformity with Ventresca (2012) and Edalatmanesh (2013), the values reported in Table 2 are the $C_3 = \sum_{i=1}^{T} \binom{|P_i|}{2}$ values rather than C_1. As a note, the K values in Ventresca (2012) for *FF250, FF500, FF1000* and *FF2000* were incorrectly tabulated and should be those shown in Table 2 *(M.Ventresca, Private Communication)*.

For one of the largest graphs (*ER2344*, 2344 nodes) used in Ventresca (2012), over 30 trials, random sampling obtained a minimum C_3 of 2,021,146, simulated annealing obtained a minimum C_3 of 2,011,122 and PBIL gave a minimum C_3 of 2,009,132 (a reduction of 0.6 % on random sampling). For this graph, CNA1 obtained the C_3 value of 1,014,430 in all 100 trials with an average CPU time of $1,094.3$ seconds (as compared to the PBIL time of 2171 seconds). As regards the variability of results for *ER2344*, PBIL obtained an average value over the 30 trials of 2,024,226.8, a minimum value of 2,009,132 and a maximum value of 2,033,234 with a standard deviation of 6365.2. All the results reported in Ventresca (2012) have a similar dispersion of results to that obtained for *ER2344*. For each problem in Table2 CNA1 always obtained the C_3 value listed in Table 2 over 100 consecutive trials.

Also shown in Table 2 are the CPU times for SA, PBIL, DFSH-post and CNA1 but comparisons are only meaningful when a lower C_3 is attained at a lower CPU time (ignoring differences in computer performance). Accordingly DFSH-post and CNA1 would both be ranked as having better performance than SA or PBIL but no ranking of CNA1 as compared to DFSH-post is possible. For example, for ER2344, CNA1 attained 1,606,773 in 0.17 CPU seconds as compared to DFSH-post taking 0.049 to attain 1,606,656. It took another 1,094 CPU seconds for CNA1 to attain the final value of 1,014,430.

An important difference between CNA1 and DFSH-post is the determination of constants or graph model specific weights. CNA1 has very few constants and these

Table 1 Comparative C_1 results for IP, CNP1c-M, and CNA1 on small sparse real-world graphs

K	C_1^M	IP	CNP1c-M	CNA1		K	C_1^M	IP	CNP1c-M	CNA1	
		CPU	CPU	C_1^T	CPU			CPU	CPU	C_1^T	CPU
Zachary's Karate Club Social network ($\|V\| = 34$, $\|E\| = 78$, $\beta = 2.29$)											
1	64.35	3.6	0.3	64.3494	0.0	2	50.98	43.1	2.8	50.9804	0.0
3	35.65	31.1	1.6	35.6506	0.0	4	14.80	9.5	0.2	14.7950	0.0
5	8.02	1.3	0.1	8.0214	0.0						
Chesapeake Bay Mesohaline Network ($\|V\| = 39$, $\|E\| = 170$, $\beta = 2.29$)											
1	94.87	155.1	13.8	94.8718	0.0	2	89.88	662.8	21.9	89.8785	0.0
3	80.30	885.9	21.4	80.2969	0.0	4	71.26	266.8	11.9	71.2551	0.1
5	59.51	497.7	16.1	59.5142	2.1						
Dolphins Social Network ($\|V\| = 62$, $\|E\| = 170$, $\beta = 2.74$)											
1	90.48	4,363.7	15.6	90.4812	0.0	2	81.44	9,667.6	30.1	81.4384	0.0
3	75.67	24,154.6	55.7	75.6742	0.0	4	46.32	3,365.1	3.8	46.3247	1.0
5	40.77	4,477.4	6.5	40.7721	0.1	6	37.33	>50, 000	20.0	37.3347	0.0
7	33.95	42,315.0	32.6	33.9503	0.0	8	30.51	>50, 000	36.7	30.5130	0.1
9	27.29	>50, 000	76.3	27.2871	0.0	10	24.27	>50, 000	78.3	24.2729	0.0

C_1^M values and IP / CNP1c-M results are from Veremyev et al. (2014). For these graphs, a trial of CNA1 terminated when it found a C_1 result that was less than or equal to $C_1^T + 0.00005$ and 100 consecutive successful CNA1 trials were required for each graph and K value

Table 2 Comparative $C_3 = \sum_{i=1}^{T} \binom{|P_i|}{2}$ results for Random Sampling (Ventresca 2012), SA (Ventresca 2012), PBIL (Ventresca 2012), DFSH-post (Edalatmanesh 2013) and CNA1 on model graphs

Graph	K	Random Min	SA Min	SA CPU	PBIL Min	PBIL CPU	DFSH-post	DFSH-post CPU	CNA1	CNA1 CPU
ER235	50	8,106	7700	38	6700	84	1141	0.004	297	0.2
ER466	80	49,391	48,627	110	44,255	183	19,952	0.013	1,548	38.4
ER941	140	238,498	234,479	361	229,576	469	114,166	0.017	5,345	146.6
ER2344	200	2,021,146	2,011,122	1,931	2,009,132	2,171	1,606,656	0.049	1,014,430	1094.3
BA500	50	1627	997	66	892	126	203	0.007	195	0.0
BA1000	75	7051	3,770	172	3,057	264	580	0.013	558	0.6
BA2500	100	57,958	31,171	840	28,044	1,178	4,292	0.031	3704	0.6
BA5000	150	353,766	170,998	3,154	146,753	3515	12,273	0.064	10,196	2.9
WS250	70	15,066	14,251	70	13,786	135	16,110	0.007	4,465	246.6
WS500	125	59,118	54,201	173	53,779	263	55,163	0.01	2,141	109.0
WS1000	200	312,481	311,700	548	308,596	676	319,600	0.02	171,635	370.1
WS1500	265	713,747	717,369	1,816	703,241	2,064	653,015	0.029	14,461	213.2
FF250	50	3,463	1841	37	1,386	88	302	0.009	194	7.2
FF500	110	3,732	2,397	156	1,904	233	344	0.014	257	7.1
FF1000	150	125,680	92,800	4,10	59,594	509	1,880	0.022	1,260	68.6
FF2000	200	609,883	387,248	1,723	256,905	1961	7,432	0.035	4,548	214.7

For these graphs, a trial of CNA1 terminated when it found the result that was less than or equal to the result listed and 100 consecutive successful CNA1 trials were required

were determined across a benchmark containing a number of different graph models using a simple mechanism (small, medium, large values for each constant over trials on Benchmark 1). These CNA1 constant values are used for all graphs in this study. However, the four weights and threshold θ value of DFSH-post for each graph model need to be tuned experimentally using generated benchmark suites containing a large number of graphs for each graph model (Erdos–Renyi, Barabasi–Albert, Watts–Strogatz, Forest Fire). As well as being a time consuming process this also limits the application of DFSH-post to model graphs where the benchmark suites can be generated for determining the four weights and threshold θ value. In contrast, CNA1 is applicable to both model and real world graphs.

3.2 Benchmark 1

Tables 3 and 4 show the comparative results attained by CNP1c-M, CNA1, CNP2c-M and CNA2 for the graphs studied in Veremyev et al. (2014). For these graphs, a trial of CNA1 terminated when it found a result that was less than or equal to $C_1^T + 0.00005$ and 100 consecutive successful CNA1 trials were required. For CNA2, a trial terminated when C_2^T was achieved and 100 consecutive successful CNA2 trials were also required. Table 5 shows the CNP1 instances in Tables 3 and 4 where the trials target (C_1^T) differs from the best found (C_1^B) by CNA1.

With regard to the results shown in Tables 3 and 4, the CNA1 results were for a C_1^T less than or equal to $C_1^M + 0.005$ (less than or equal to $C_1^M + 0.05$ for *bcspwr03* and *USAir97*) and all 100 trials of all instances were within 0.00005 of C_1^T with the following exceptions:

- Instance *bcspwr07* with $K = 300$ where C_1^T exceeded $C_1^M + 0.005$ by 0.0011 in the 100 trials.
- Instance *USAir97* with $K = 25$ where C_1^T exceeded $C_1^M + 0.05$ by 1.0199 . There is no obvious reason for this *USAir97* discrepancy for CNA1 but it may be related to the high β value for this graph.
- The discrepancy between the C_1^M and C_1^T results for *Erdos971* arise because this graph, as downloaded from the University of Florida Sparse Matrix Collection, has a number of components. In Veremyev et al. (2014) the results were attained using only the largest component ($|V| = 429, |E| = 1,312$) as a starting point. In contrast, CNA1/CNA2 used the complete graph for *Erdos971*.

Of note in Table 3 is the *bcspwr03*, $K = 4$ result where the CNA1 C_1^T result was considerably less than the C_1^M value. As $K < 5$, the CNA1 C_1^T value was the result of CNA1 performing a complete search.

As is well known, heuristic algorithms such as CNA1 initially approach the target very quickly but take a long time for the final steps. For example, a trial on *494_Bus* with $K = 100$ and CPU time close to the average for the 100 trials, reduced C_1 by 99.66 % of the reduction required in the first 0.31 % of the CPU time and then used the remaining 99.69 % of CPU time to make the final 0.34 % (0.000796) reduction required in C_1 to achieve the target of 0.2324. Clearly, if $C_1^M + 0.005$ had been used as the target for CNA1, this trial would have completed in a much reduced CPU time. Overall, the effect of reducing the accuracy by using C_1^M as the target C_1 value for

Table 3 Benchmark 1a: comparative results for algorithms CNP1c-M / CNA1 and CNP2c-M / CNA2 on medium-size sparse real-world graphs

K	CNP1c-M		CNA1		L	CNP2c-M		CNA2	
	C_1^M	CPU	C_1^T	CPU		C_2^M	CPU	C_2^T	CPU
bcspwr03 ($\|V\| = 118$, $\|E\| = 179$, $\beta = 1.52$)									
2	54.4	23.1	54.3966	0.0	2	41	0.2	41	0.0
3	48.3	157.9	48.3268	0.9	3	33	0.4	33	0.0
4	42.5	50.9	34.6516	2.3	4	27	0.8	27	0.0
5	28.5	78.3	28.4659	0.1	5	26	21.5	26	0.0
6	23.7	75.2	23.7433	0.0	6	23	24.1	23	0.0
7	19.4	60.2	19.3829	0.2	7	20	12.4	20	0.0
8	15.5	81.8	15.5150	0.0	8	19	45.9	19	0.0
9	12.6	61.9	12.6032	0.0	9	16	14.7	16	0.0
10	10.3	41.6	10.2709	0.0	10	15	20.9	15	0.0
bcspwr04 ($\|V\| = 274$, $\|E\| = 669$, $\beta = 2.44$)									
100	0.23	784.1	0.2299	0.1	2	118	6.5	118	0.0
110	0.14	660.4	0.1444	0.1	3	97	378.5	97	0.5
120	0.08	130.4	0.0829	0.1	4	–	>50,000	89	0.0
130	0.05	21.5	0.0455	0.0	5	–	>50,000	80	0.9
USAir97 ($\|V\| = 332$, $\|E\| = 2,126$, $\beta = 6.40$)									
10	47.0	6059.1	47.0389	0.1	2	115	7.6	115	0.1
15	31.5	10, 343.1	31.4582	0.8	3	96	1,420.6	96	0.1
20	21.0	7016.1	20.9733	1.9	4	86	12,855.0	86	0.0
25	14.6	15, 597.3	15.6699	0.1	5	–	>50,000	80	0.1
30	10.5	20, 447.4	10.5158	15.9	6	–	>50,000	73	0.0
bcspwr05 ($\|V\| = 443$, $\|E\| = 590$, $\beta = 1.33$)									
50	1.27	1055.6	1.2676	2.4	2	143	1.1	143	0.2
60	0.85	175.6	0.8488	0.9	3	111	1.9	111	0.0
70	0.60	51.4	0.6047	0.4	4	96	28.8	96	0.0
80	0.45	49.8	0.4525	0.5	5	83	155.0	83	17.6
90	0.34	31.1	0.3361	0.6	6	73	4245.8	73	0.3
100	0.25	30.8	0.2503	85.3	7	65	6838.7	65	0.2
bcspwr06 ($\|V\| = 1,454$, $\|E\| = 1,923$, $\beta = 1.32$)									
300	0.10	2727.4	0.0969	127.5	2	460	9.9	460	8.4
330	0.07	1092.8	0.0742	76.6	3	371	183.9	371	19.4
370	0.05	1082.1	0.0528	1008.2	4	–	>50,000	310	32.9
400	0.04	1021.5	0.0404	39.6	5	–	>50,000	276	258.6
bcspwr07 ($\|V\| = 1,612$, $\|E\| = 2,106$, $\beta = 1.31$)									
300	0.10	2838.6	0.1058	2850.0	2	496	10.8	496	10.5
330	0.08	1955.7	0.0827	1088.6	3	404	55.4	404	8.3
370	0.06	1091.0	0.0595	77.6	4	–	>50,000	340	19.9
400	0.05	1692.1	0.0465	124.9	5	–	>50,000	294	22.7

All CNA1/CNA2 CPU results are the average over 100 consecutive successful trials with $C_1 \leq C_1^T + 0.00005$ for CNA1 and $C_2 = C_2^T$ for CNA2

Table 4 Benchmark 1b: Comparative results for algorithms CNP1c-M / CNA1 and CNP2c-M / CNA2 on medium-size sparse real-world graphs

K	CNP1c-M		CNA1		L	CNP2c-M		CNA2	
	C_1^M	CPU	C_1^T	CPU		C_2^M	CPU	C_2^T	CPU
Erdos971 ($\|V\| = 472, \|E\| = 1,314, \beta = 2.78$)									
125	0.21	4993.4	0.1745	24.3	2	159	7.9	159	0.4
130	0.17	902.7	0.1457	1.1	3	135	3427.6	135	0.3
135	0.15	622.0	0.1224	0.8	4	–	>50,000	120	0.7
140	0.13	331.5	0.1053	1.3	5	–	>50,000	112	7.1
145	0.11	277.4	0.0918	0.2	6	–	>50,000	105	1.0
celegans_metabolic ($\|V\| = 453, \|E\| = 2,025, \beta = 4.47$)									
100	0.44	2899.1	0.4444	0.2	2	173	20.8	173	0.1
125	0.25	140.2	0.2510	0.3	3	135	48,156.2	135	0.7
150	0.15	341.7	0.1534	0.1	4	–	>50,000	116	0.1
175	0.09	94.8	0.0879	0.1	5	–	>50,000	102	0.1
494_Bus ($\|V\| = 494, \|E\| = 584, \beta = 1.19$)									
50	0.94	99.9	0.9378	0.2	2	151	0.8	151	0.2
60	0.65	36.0	0.6488	0.6	3	112	1.1	112	0.1
70	0.47	32.7	0.4730	0.3	4	93	2.8	93	0.1
80	0.37	30.2	0.3728	2.1	5	81	21.0	81	2.5
100	0.23	35.2	0.2324	22.6	6	69	27.9	69	0.6
662_Bus ($\|V\| = 662, \|E\| = 906, \beta = 1.37$)									
200	0.08	145.5	0.0763	6.8	2	215	3.1	215	1.5
220	0.05	162.1	0.0507	1.7	3	172	11.3	172	0.2
230	0.04	160.5	0.0416	295.6	4	144	110.6	144	1.0
250	0.03	151.9	0.0283	0.4	5	–	>50,000	128	1.2
1138_Bus ($\|V\| = 1,138, \|E\| = 1,458, \beta = 1.28$)									
100	0.52	2,859.1	0.5212	14.4	2	331	1,541.1	331	2.0
150	0.24	651.3	0.2384	20.3	3	261	>50,000	261	0.9
200	0.13	595.4	0.1288	892.0	4	218	>50,000	218	0.7
250	0.07	360.4	0.0736	15.3	5	184	>50,000	184	2.7
300	0.04	371.9	0.04453	745.6	6	–	>50,000	167	23.3

For these graphs, a trial of CNA1 terminated when it found a result that was less than or equal to $C_1^T + 0.00005$ and 100 consecutive successful CNA1 trials were required. For CNA2, a trial terminated when C_2^T was achieved and 100 consecutive successful CNA2 trials were also required

CNA1 and deeming a trial of CNA1 as successful if it attained a C_1 value less than or equal to $C_1^M + 0.005$ ($C_1^M + 0.05$ for *bcspwr03* and *USAir*) reduced the total CNA1 CPU time to complete the benchmark by approximately 85 % of that shown. This choice of testing CNA1 to four decimal places was to demonstrate the accuracy with which CNA1 is able to repeat results and also to create a more demanding benchmark for the future.

Table 5 CNP1 instances in Tables 3 and 4 where the trials target (C_1^T) differs from the best found (C_1^B) by CNA1

Graph	K	C_1^M	C_1^T	C_1^B	Graph	K	C_1^M	C_1^T	C_1^B
1138_Bus	200	0.13	0.1288	0.1286	bcspwr06	300	0.10	0.0969	0.0965
bcspwr06	330	0.07	0.0742	0.0739	bcspwr07	300	0.10	0.1058	0.1056
bcspwr07	370	0.06	0.0595	0.0591					

CNA2 achieved all C_2^M values attained in Veremyev et al. (2014). As C_2^M is an integer, there were no rounding issues associated with the interpretation of these results.

An indication of the difficulty of problem CNP1 as compared to that of problem CNP2 in this study can be gained by comparing C_1^T for the solution of a CNP1 problem with the C_1 value attained for an equivalent CNP2 problem. From Table 4, for *Erdos971* with $K = 135$, C_1^T was set at 0.1224. However, the CNA2 solution for *Erdos971* with $L = 3$, $K = 135$ has a C_1 value of 0.1394 indicating that, for this instance at least, CNP2 is a less demanding problem than CNP1. Given the relative CPU times required by CNA1 and CNA2 and the similarity between these two algorithms, this is probably true for all instances in this study.

3.3 Benchmark 2

The graphs for Benchmark 2 are sparse real-world graphs from Davis and Hu (2011) that have more nodes or a higher β than those used in Benchmark 1. The limit on the size of these graphs and the values used for C_1^T/C_2^T was determined on the basis that it was practical to perform 100 consecutive successful trials of CNA1/CNA2 for each graph with the results shown in Table 6.

For these graphs, as there are no exact results available for comparison, each problem was addressed by, for each instance, performing a long-running experiment to identify the best result C_1^B/C_2^B that CNA1/CNA2 could attain. From this, a repeatable trial target C_1^T/C_2^T that could be attained in reasonable time for 100 consecutive successful trials with different starting initialisations was identified. Table 6 shows C_1^T/C_2^T for repeatable trial results, Table 7 shows C_1^B where it differs from the repeatable trial target C_1^T and Table 8 shows C_2^B where it differs from the repeatable trial target C_2^T.

3.4 CNA1 performance

The distributions of C_1 values obtained by executing CNA1 for a fixed number of node selections over 100 trials are shown in Fig. 1 for *1138_Bus*, $K = 300$, $C_1^T = 0.0445$ (Table 4), *bcspwr09*, $K = 400$, $C_1^T = 0.0607$ (Table 6), *email*, $K = 400$, $C_1^T = 0.1039$ (Table 6) and *power*, $K = 400$, $C_1^T = 0.2116$ (Table 6). While C_1 is a float number, it is the ratio of two integers and, for 100 trials, at most 100 C_1 values will occur. In some cases, there are just a few valid solutions near the best solution thus

Table 6 Benchmark 2: Repeatable results for algorithms CNA1 and CNA2 on larger sparse real-world graphs

K	CNA1		K	CNA1		L	CNA2		L	CNA2	
	C_1^T	CPU		C_1^T	CPU		C_2^T	CPU		C_2^T	CPU
Erdos991 ($\|V\| = 429, \|E\| = 1,417, \beta = 2.88$)											
50	18.9881	238.0	60	11.0245	201.0	2	168	0.2	3	142	0.1
70	5.4576	123.9	80	2.3761	143.1	4	128	0.5	5	119	2.7
90	1.1516	62.4				6	114	0.5			
Erdos981 ($\|V\| = 485, \|E\| = 1,381, \beta = 2.85$)											
50	18.3565	242.5	60	10.4959	171.5	2	165	0.3	3	139	0.3
70	4.6503	197.2	80	2.1349	67.9	4	126	0.2	5	118	0.2
90	1.0326	40.8				6	112	9.4			
email ($\|V\| = 1,133, \|E\| = 5,451, \beta = 4.81$)											
300	1.2882	928.2	330	0.5355	862.4	2	494	2,890.1	3	444	210.7
370	0.1820	785.0	400	0.1039	941.7	4	418	130.8	5	397	203.3
bcspwr08 ($\|V\| = 1,624, \|E\| = 2,213, \beta = 1.36$)											
300	0.1125	214.3	330	0.0877	343.9	2	517	11.5	3	409	53.0
370	0.0633	142.7	400	0.0507	154.6	4	349	19.5	5	309	279.8
bcspwr09 ($\|V\| = 1,723, \|E\| = 2,394, \beta = 1.39$)											
300	0.1409	211.0	330	0.1087	216.2	2	556	21.8	3	438	15.1
370	0.0781	164.6	400	0.0607	228.4	4	382	32.1	5	340	300.3
yeast ($\|V\| = 2,361, \|E\| = 6,646, \beta = 2.82$)											
300	0.7910	1,327.3	330	0.3256	806.3	2	618	63.2	3	539	13.6
370	0.1334	823.3	400	0.0828	1,545.4	4	489	262.9	5	457	1,637.0
power ($\|V\| = 4,941, \|E\| = 6,594, \beta = 1.34$)											
300	0.3871	315.4	330	0.3137	394.3	2	1522	589.4	3	1200	2381.5

Table 6 continued

K	CNA1		K	CNA1		L	CNA2		L	CNA2	
	C_1^T	CPU		C_1^T	CPU		C_2^T	CPU		C_2^T	CPU
370	0.2489	754.4	400	0.2116	698.4	4	1001	767.6	5	880	2209.2
bcspwr10 ($\|V\| = 5,300$, $\|E\| = 8,271$, $\beta = 1.56$)											
400	0.7868	587.3	430	0.6581	641.2	2	1970	1794.9	3	1623	1164.3
470	0.5382	673.7	500	0.4600	839.7	4	1430	7109.8	5	1282	8261.4
Erdos972 ($\|V\| = 5,488$, $\|E\| = 7,085$, $\beta = 1.29$)											
175	0.5201	1893.1	180	0.4015	842.7	2	417	34.2	3	397	4.6
185	0.3172	1,095.5	190	0.2575	1,209.0	4	379	4.9	5	363	4.7
195	0.2072	1,515.3				6	347	4.9			
Erdos982 ($\|V\| = 5,822$, $\|E\| = 7,375$, $\beta = 1.27$)											
155	3.1060	1,268.9	160	2.2387	1617.0	2	430	42.3	3	410	5.5
165	1.4934	1,424.6	170	0.9699	1533.6	4	391	5.8	5	376	5.6
175	0.6742	1,771.8				6	358	8.8			
ca-HepTh ($\|V\| = 9,877$, $\|E\| = 25,973$, $\beta = 2.63$)											
500	25.1874	205.3	530	24.0000	126.5	2	3340	2148.6	3	2660	3189.3
570	20.3373	250.9	600	18.2668	304.2	4	2271	2801.1	5	2003	3749.9

For these graphs, a trial of CNA1 terminated when it found a result that was less than or equal to $C_1^T + 0.00005$ and 100 consecutive successful CNA1 trials were required. For CNA2, a trial terminated when C_2^T was achieved and 100 consecutive successful CNA2 trials were also required. Where a CPU result is presented as $-$, CNA2 was unable to repeat the C_2^T result over 100 consecutive trials

Table 7 CNP1 instances in Table 6 where the repeatable trials target (C_1^T) differs from the best found value (C_1^B) for CNA1

Graph	K	C_1^T	C_1^B	Graph	K	C_1^T	C_1^B
Erdos991	50	18.9881	18.3597	Erdos991	60	11.0245	10.5401
Erdos991	70	5.4576	4.9310	Erdos991	80	2.3761	2.3430
Erdos991	90	1.1516	1.1417	Erdos981	50	18.3565	17.8819
Erdos981	60	10.4959	9.9898	Erdos981	70	4.6503	4.3734
Erdos981	80	2.1349	2.1258	bcspwr08	300	0.1125	0.1120
bcspwr08	330	0.0877	0.0875	bcspwr08	370	0.0633	0.0630
bcspwr08	400	0.0507	0.0504	bcspwr09	300	0.1409	0.1397
bcspwr09	330	0.1087	0.1079	bcspwr09	370	0.0781	0.0777
bcspwr09	400	0.0607	0.0606	Erdos972	175	0.5201	0.5064
Erdos972	180	0.4015	0.3911	Erdos972	185	0.3172	0.3113
Erdos972	190	0.2575	0.2529	Erdos972	195	0.2072	0.2049
Erdos982	155	3.1060	2.6677	Erdos982	160	2.2387	2.0732
Erdos982	165	1.4934	1.2673	Erdos982	170	0.9699	0.9010
Erdos982	175	0.6742	0.6221	email	300	1.2882	1.2516
email	330	0.5355	0.4761	email	370	0.1820	0.1812
email	400	0.1039	0.1034	yeast	300	0.7910	0.7321
yeast	330	0.3256	0.3098	yeast	370	0.1334	0.1318
yeast	400	0.0828	0.0822	power	300	0.3871	0.3795
power	330	0.3137	0.3105	power	370	0.2489	0.2462
power	400	0.2116	0.2104	bcspwr10	400	0.7868	0.7623
bcspwr10	430	0.6581	0.6405	bcspwr10	470	0.5382	0.5262
bcspwr10	500	0.4600	0.4554				

Table 8 CNP2 instances in Table 6 where the repeatable trials target (C_2^T) differs from the best found value (C_2^B) for CNA2

Graph	L	C_2^T	C_2^B	Graph	L	C_2^T	C_2^B
email	5	397	396	yeast	4	489	488
power	2	1522	1521	power	5	880	878
bcspwr10	2	1970	1968	bcspwr10	3	1623	1621
bcspwr10	4	1430	1425	bcspwr10	5	1282	1274
ca-HepTh	3	2260	2659	ca-HepTh	4	2271	2270
ca-HepTh	5	2003	2001				

giving a small number of possible values for C_1 in the distribution (C_1 is to an accuracy of four decimal places). For example, for the *Erdos* family of graphs all trials attained the same C_1 value while for *1138_Bus* with $K = 300$ there were only two C_1 values attained over the 100 trials and just six distinct C_1 values for *bcspwr09*, $K = 400$. For the *email* graph with $K = 400$ there were more C_1 values with the minimum C_1

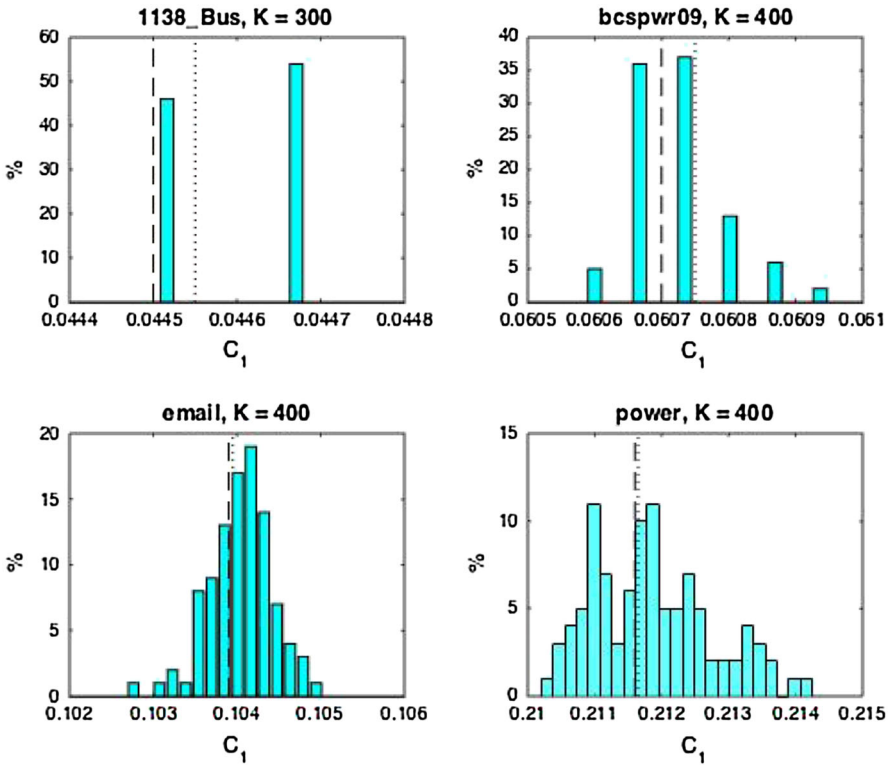

Fig. 1 Distribution of C_1 values attained over 100 trials of CNA1 where each trial was terminated after a fixed number of node selections. The dashed line shows the value used for C_1^T and the dotted line shows $C_1^T + 0.00005$ when performing the CNA1 benchmark trials for each of these graphs

value attained being 0.102764, a maximum C_1 of 0.104947 and a standard deviation of 0.000382459. For the *power* graph with $K = 400$ there was a large number of different C_1 values attained with a minimum value of 0.210254, a maximum value of 0.214236 and a standard deviation of 0.000886934 over the 100 trials. These relatively low standard deviation figures demonstrate the consistency of CNA1 in finding good solutions.

The CNA1 Run Length Distributions (RLDs) for four sets of 100 consecutive successful trials for graphs for *1138_Bus*, $K = 300$, *bcspwr09*, $K = 400$, *email*, $K = 400$ and *power*, $K = 400$ are shown in Fig. 2. 0.0445 and 0.0447). For *1138_Bus*, $K = 300$ to attain a C_1^T of 0.0447 CNA1 takes, on average, 0.8754 CPU seconds while to attain 0.0445 takes on average 3, 745.6 CPU seconds. As can be seen from Fig. 2, 50 % of *1138_Bus*, $K = 300$ trials complete very quickly while the other 50 % take much longer however all these trials follow the C_1 progression 0.0447 to 0.0445. The other RLD results are typical of heuristic algorithms where some trials complete quickly while a small proportion take considerably longer.

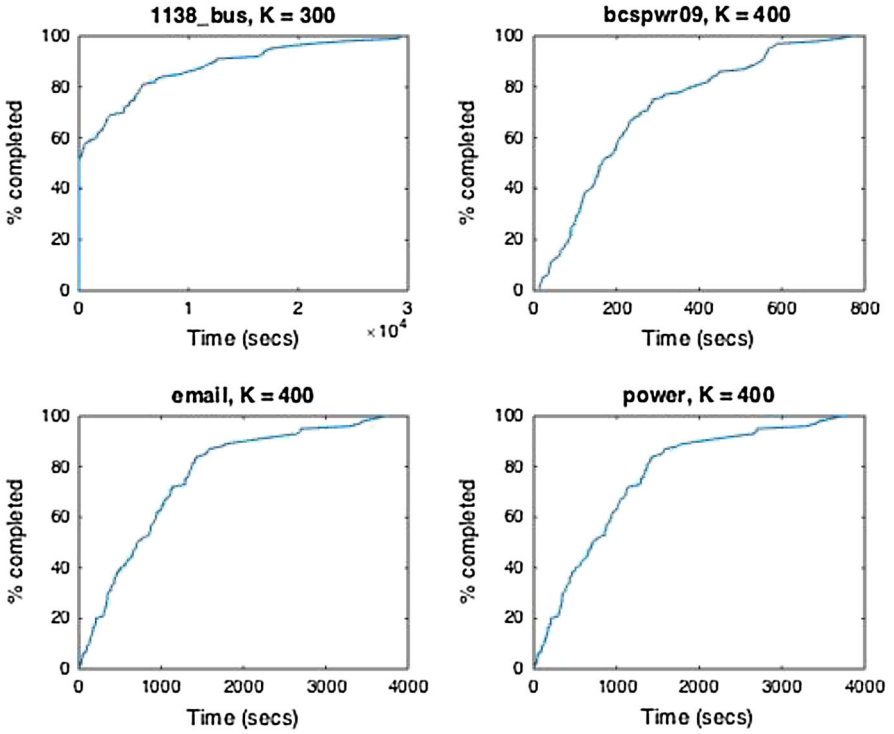

Fig. 2 Run length distributions for CNA1 showing the rate at which the benchmark trials were successfully completed

4 Conclusion

In this paper, two new heuristic algorithms were proposed for two variants of the critical node detection problem. The distinguishing feature of both these algorithms is that they operate at the graph component level which leads to efficiencies when the impacts of node additions and removals are evaluated. The results obtained by these algorithms were evaluated against exact solutions recently obtained on sparse real-world graphs containing up to 1, 500 nodes. In addition, new results were also presented for larger sparse real-world graphs containing up to 9, 877 nodes and 25, 793 edges.

With regard to these new algorithms, future research directions include enhancing their efficiency as well as targeting less sparse and larger graphs, model graphs, and weighted graphs. In addition, other variants of the critical node detection problem will also be addressed.

Acknowledgments The author would like to thank Andrea Grosso, Dipartimento di Informatica, Università degli Studi di Torino, for discussions on the critical node detection problem. In addition the author would like to thank the anonymous referees whose questions and suggestions resulted in a considerable improvement in this paper.

References

Addis, B., Di Summa, M., Grosso, A.: Identifying critical nodes in undirected graphs: complexity results and polynomial algorithms for the case of bounded tree width. Discret. Appl. Math. **161**(16), 2349–2360 (2013)

Arulselvan, A., Commander, C.W., Pardolas, P.M., Shylo, O.: Managing network risk via critical node identification. In: Gülpinar, N., Rüstem, B. (eds.) Risk Management in Communications Systems. Springer-Verlag, Hiedelburg (2009)

Arulselvan, A., Commander, C., Shylo, O., Pardolas, P.: Cardinality-constrained critical node detection problem. In: Gülpinar, N., Harrison, P., Rüstem, B. (eds.) Performance Models and Risk Management in Communications Systems, pp. 79–91. Springer-Verlag, Hiedelburg (2011)

Arulselvan, A., Commander, C.W., Elefteriadou, L., Pardolas, P.M.: Detecting critical nodes in sparse graphs. Comput. Oper. Res. **36**, 2193–2200 (2009)

Boginski, V., Commander, C.W.: Identifying critical nodes in protein-protein interaction networks. In: Butenko, S., Chaovilitwongse, W.A., Pardalos, P.M. (eds.) Clustering Challenges in Biological Networks, pp. 153–166. World Scientific Publishing Co., New Jersey, USA (2008)

Borgatti, S.P.: Identifying sets of key players in a social network. Comput. Math. Organ. Theory **12**(1), 21–34 (2006)

Davis, T.A., Hu, Y.: The university of florida sparse matrix collection. ACM Trans. Math. Softw. **38**, 1–25, http://www.cise.ufl.edu/research/sparse/matrices/ (2011)

Di Summa, M., Grosso, A., Locatelli, M.: Complexity of the critical node problem over trees. Comput. Oper. Res. **38**(12), 1766–1774 (2011)

Di Summa, M., Grosso, A., Locatelli, M.: Branch and cut algorithms for detecting critical nodes in undirected graphs. Comput. Optim. Appl. **53**(3), 649–680 (2012)

Dinh, T., Xuan, Y., Thai, M., Park, E., Znati, T.: On approximation of new optimisation methods for assessing network vulnerability. In: INFOCOM, 2000 Proceedings of the IEEE, pp. 1–9 (2010)

Dinh, T., Xuan, Y., Thai, M., Pardalos, P., Znati, T.: On new approaches of assessing network vulnerability: hardness and approximation. IEEE/ACM Trans. Netw. **20**(2), 609–619 (2012)

Dinh, T., Thai, M.T., Nguyen, H.T.: Bound and exact methods for assessing link vulnerability in complex networks. J Comb. Optim. **28**(1), 3–24 (2014)

Edalatmanesh, M.: Heuristics for the critical node detection problem in large complex networks. Brock University, St. Catharines, Ontario, vol. 6, http://www.dr.library.brocku.ca/bitstream/handle/10464/4984/Brock_Edalatmanesh_Mahmood_2013.pdf?sequence=1 (2013)

Fan, N., Pardalos, P.M.: Robust optimisation of graph partitioning and critical node detection in analysing networks, In: Wu, W., Daescu, O. (eds.) Combinatorial Optimization and Applications, vol. 6508 of Lecture notes in computer science, pp. 170–183. Springer, Berlin (2010)

Krebs, V.: Uncloaking terrorist networks, http://www.firstmonday.org/issues/issue7_4/krebs/ (2002)

Medlock, J., Galvani, A.P.: Optimizing influenza vaccine distribution. Science **325**, 1705–1708 (2009)

Nguyen, D.T., Shen, M.T., Thai, M.T.: Detecting critical nodes in interdependent power networks for vulnerability assessment. IEEE Trans. Smart Grid **99**, 1–9 (2013)

Shen, Y., Dinh, T.N., Thai, M.T.: Adaptive algorithms for detecting critical links and nodes in dynamic networks. In: Proceedings of the IEEE Military Communications Conference–MILCOM (2012)

Shen, Y., Nguyen, N.P., Xuan, Y., Thai, M.T.: On the discovery of critical links and nodes for assessing network vulnerability. IEEE/ACM Trans. Netw. **21**(3), 963–973 (2013)

Sun, F., Shayman, M.A.: On pairwise connectivity of wireless multi hop networks. Int. J. Secur. Netw. **2**(1/2), 37–49 (2007)

Ventresca, M., Aleman, D.: A fast greedy algorithm for the critical node detection problem. In: Zhang, Z., Wu, L., Xu, W., Du, D.-Z. (eds.) Combinatorial Optimisation and Applications, vol. 8851 of Lecture notes in computer science, pp. 613–624. Springer (2014)

Ventresca, M., Aleman, D.: A region growing algorithm for detecting critical nodes. In: Zhang, Z., Wu, L., Xu, W., Du, D.Z. (eds.) Combinatorial Optimisation and Applications, vol. 8851 of Lecture notes in computer science, pp. 593–612. Springer (2014)

Ventresca, M., Aleman, D.: Approximation algorithms for detecting critical nodes, In: NATO Science for Peace and Security Series-D: Information and Communication Security, pp. 289–305, IOS Press (2014)

Ventresca, M.: Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem. Comput. Oper. Res. **39**, 2763–2775 (2012)

Ventresca, M., Aleman, D.: A derandomized approximation algorithm for the critical node detection problem. Comput. Oper. Res. **43**, 261–270 (2014)
Veremyev, A., Boginski, V., Pasiliao, E.L.: Exact identification of critical nodes in sparse networks via new compact formulations. Optim. Lett. **8**(1), 1245–1259 (2014)