Computers & Operations Research 39 (2012) 2763–2775

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem

Mario Ventresca

Medical Operations Research Laboratory, Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada

ARTICLE INFO

Available online 17 February 2012

Keywords:
Critical node detection
Complex networks
Random graph
Simulated annealing
Population-based incremental learning

ABSTRACT

In this paper the problem of critical node detection (CNDP) is approached using population-based incremental learning (an estimation of distribution algorithm) and simulated annealing optimization algorithms using a combinatorial unranking-based problem representation. This representation is space-efficient and alleviates the need for any repair mechanisms. CNDP is a very recently proposed problem that aims to identify a vertex set $V' \subseteq V$ of $k > 0$ nodes from a given graph $G = (V,E)$ such that $G(V \backslash V')$ has minimum pairwise connectivity. Numerous practical applications for this problem exist, including pandemic disease mitigation, computer security and anti-terrorism. In order to test the proposed heuristics 16 benchmark random graph structures are additionally proposed that utilize Erdos–Renyi, Watts–Strogatz, Forest Fire and Barabasi–Albert models. Each of these models presents different network characteristics, yielding variations in problem difficulty. The relative merits of the two proposed approaches are compared and it is found that the population-based incremental learning approach, using a windowed perturbation operator is able to outperform the proposed simulated annealing method.

1. Introduction

The critical node detection problem is $\mathcal{NP}$-complete and concerned with discovering a set of k vertices $S \subseteq V$ of graph $G(V,E)$ whose removal will leave the graph with minimum pairwise connectivity [1]. As a consequence of sufficiently large k, the resulting graph $G' = G(V \backslash S, E \backslash \{(i,j) : i,j \notin V \backslash S\})$ will have the maximum number of components, while simultaneously having the minimum variance in component sizes. In general, identification of critical nodes provide important information concerning the structural characteristics of the graph under consideration.

Let there exist a diffusive process acting upon the network, for instance pandemic disease spread or information transfer. Where an edge is present between two vertices $(i,j) \in E$, a path exists for the diffusive process to propagate through the network. By removing critical nodes we can limit the amount of spread over the entire network. Not surprisingly, CNDP has applications in network immunization where vaccination/quarantine of individuals is expensive. Existing approaches for vaccination in these circumstances focus on using properties of node centrality, however these are not optimal strategies [1]. Other applications include transportation engineering, protein–protein interaction networks [2], disease mitigation [3] and computer security [4,5], for example.

As stated above, the CNDP problem has only recently been proposed and therefore there does not exist a great variety of approaches to solving this problem. The original works [1,2,5] are based on heuristic algorithms and are shown to yield reasonable results. However, in [2] a simple genetic algorithm was proposed and shown to outperform the heuristic method with respect to solution quality. Another caveat of these approaches is the use of networks that contain less than 200 nodes; for networks with thousands of vertices the heuristic is too computationally expensive.

A further analysis of graph partitioning and critical node detection was performed in [6]. The work considers the concept of uncertain edges and describes the effects on community structure that result. Numerical experiments are performed to verify the theoretical analysis. The cardinality constrained critical node detection problem is proposed in [7]. This version extends CNDP by restricting the disconnected components to a given cardinality.

Most recently, Di Summa et al. [8] examined the case where G is restricted to a tree structure and proved the $\mathcal{NP}$-completeness of the problem for non-unit edge costs. They also provide a polynomial time dynamic programming algorithm for solving the problem with unit edge costs. The dynamic programming solution is also applied to variants of the CNDP [9]. In [10] an integer linear programming model with non-polynomial number of constraints is given and branch and cut algorithms are proposed. This latter approach also requires significant computational resources for networks containing thousands of vertices.

E-mail address: mario.ventresca@utoronto.ca

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2012.02.008

In this paper we examine the efficacy of simulated annealing and population-based incremental learning for approximating solutions to the CNDP. Neither algorithm has been applied to this problem previously. Both approaches rely upon a perturbation operator for generating successive solutions, and we propose an efficient window-based procedure. Key to the relatively low computational cost of this approach is the proposed binary solution representation. By employing a combinatorial unranking algorithm as a mapping from the binary representation to a set of removed vertices, we eliminate the need for any repair mechanism. Often repairing solutions to enforce feasibility is computationally expensive and introduces undesirable bias into the results. Moreover, generation of binary strings is very efficient as are unranking algorithms. Fitness evaluation is accomplished via a depth-first search of $\mathcal{O}(|V|+|E|)$ and thus is also not a large burden on efficiency.

In order to test the results, 16 benchmark problems are additionally proposed. These problems have been generated based on Barabasi–Albert [11], Erdos–Renyi [12], Watts–Strogatz [13] and Forest Fire [14] complex network construction algorithms. Each type of graph presents different structural properties, leading to variations in problem difficulty. We present small (less than 1000 vertices) and medium-sized (less than 5000 vertices) graphs. Extremely large networks, with millions of nodes, will be considered in future works.

The remainder of this paper is organized as follows. Section 2 formally presents the critical node detection problem and a possible integer programming formulation. The algorithms and related implementation issues are then discussed in Section 3. Section 4 proposes 16 benchmark data instances which are subsequently used in Section 5 where experimental results are discussed. Conclusions are given in Section 6.

2. Critical node detection

In this section an integer program problem formulation for the critical node detection problem (CNDP) is given. We assume graph $G=(V,E)$ is undirected and unweighted, and has vertices V and edges E. For subset $S\subseteq V$, the subgraph $G(S)$ has edge set $W=\{(i,j): i,j\in S\}\cap E$. A connected component of G is a set of vertices such that all vertices in each set are mutually connected (reachable by some path), and no two vertices in different sets are connected. Given integer $k>0$, the goal of the CNDP is to determine

$$\underset{S\subseteq V}{\operatorname{argmin}} \sum_{i,j\in(V\setminus S)} u_{ij}(G(V\setminus S)) \tag{1}$$

such that $|S|\leq k$, and where

$$u_{ij}=\begin{cases}1 & \text{if } i,j \text{ in same component of } G(V\setminus S)\\ 0 & \text{otherwise.}\end{cases} \tag{2}$$

The problem can be expressed as the following integer program [5]:

$$\text{Minimize} \quad \sum_{i,j\in V} u_{ij} \tag{3}$$

$$\text{subject to} \quad u_{ij}+v_i+v_j\geq 1 \quad \forall (i,j)\in E \tag{4}$$

$$u_{ij}+u_{jk}+u_{ki}\neq 2 \quad \forall (i,j,k)V \tag{5}$$

$$\sum_{i\in V} v_i\leq k, \tag{6}$$

$$u_{ij}\in\{0,1\} \quad \forall\, i,j\in V \tag{7}$$

$$v_i\in\{0,1\} \quad \forall\, i\in V \tag{8}$$

Optimization of this objective function will result in a graph with minimum pair-wise connectivity. The first constraint (Eq. (4)) enforces the separation of vertices into different components. Eq. (5) is concerned with a triangle inequality; if vertices i,j and j,k are in the same component, then i and k must also be in the same component. The final constraint, in Eq. (6), ensures the removal of no more than k vertices.

Eq. (2) is a measure of the pairwise connectivity of a given (sub)graph, where u_{ij} is a binary value equal to 1 iff i and j are in the same component of $G(V\setminus S,W)$. Let $\mathcal{C}$ be the set of all maximally connected components of $G(V\setminus S,W)$, and σ_c represent the size of component $c=\{1,\ldots,|\mathcal{C}|\}$. Then, an alternative objective function to Eq. (3) is [5]:

$$\frac{1}{2}\sum_{c\in\mathcal{C}}\sigma_c(\sigma_c-1) \tag{9}$$

Minimization of (9) will maximize the number of components while also minimizing the variance between components. Determining graph component sizes can be efficiently determined via a depth-first search with complexity $\mathcal{O}(|V|+|E|)$ [15] and thus does not have great computational overhead.

As indicated in [1,5] the CNDP is related to the maximum k-cut problem (K-MAX-CUT) [16]. Specifically, K-MAX-CUT approximations would be useful if only the number of components is considered. However, this additional consideration increases the problem difficulty sufficiently that related approximations are not very useful. To the best of the author's knowledge, the application of stochastic search algorithms to this problem has not yet been investigated outside of preliminary work utilizing genetic algorithms [2].

3. Algorithms and methods

In this paper we employ an indirect binary problem representation that is mapped to a set of k removed vertices. Specifically, we use an intermediate function $\phi:\{0,1\}^m\mapsto\mathbb{Z}^*$, to map an m-dimensional binary string to an integer, where a combinatorial unranking function will in turn map to a lexicographically ordered combination (i.e. $\psi:\mathbb{Z}^*\mapsto C_{|V|,k}$, where V is the vertex set and $k>0$ is user-defined). In this section the unranking function ψ is described, followed by the solution representation and stochastic search algorithms.

3.1. Solution representation

The representation being searched is a straightforward binary string. This solution representation we have implemented completely alleviates the need for repair mechanisms or penalty functions, which are often computationally expensive. Our idea utilizes a combinatorial unranking method as an intermediate step between the binary problem representation (for the search) and a lexicographically ordered (i.e. ranked) set of solutions (i.e. unique sets of vertices). The entire decoding procedure has computational complexity $\mathcal{O}(\log_2|V|+|V|)=\mathcal{O}(|V|)$, as described below.

3.1.1. Unranking

An unranking algorithm will construct elements a_i of the κth sequence $\langle a_1,\ldots,a_i,\ldots,a_k\rangle$, where $1\leq a_1<\cdots<a_i<\cdots a_k\leq|V|$ and $1\leq\kappa\leq C_{|V|,k}$. That is, given a list of lexicographically ordered combinations and an integer $\kappa>0$, the algorithm will return the associated κth combination. Characterization of unranking algorithms is according to Refs. [17,18]:

1. linear ordering of elements and
2. algorithm applied to evaluate the binomial coefficient.

The linear ordering of elements used in this paper is simply according to the natural integer value. Computation of binomial

coefficients is described below. Further information on (un)ranking algorithms can be found in [18–25].

We implement a simple unranking algorithm from [23], although more complex algorithms exist (see above unranking references for examples). Firstly, a modified Pascal's Triangle is constructed[1] as:

$$P_{i,j} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 1 & 1 & 1 & 1 & 1 & \cdots & \\ 2 & 3 & 4 & 5 & \cdots & & \\ 3 & 6 & 10 & \cdots & & & \\ 4 & 10 & \cdots & & & & \\ 5 & \cdots & & & & & \end{bmatrix} \tag{10}$$

The unranking method is presented in Algorithm 1, where ranks are assumed to be in decreasing lexicographic order. The κ-unranked combination, stored in object R, will be determined via a divide-and-conquer approach, where $|V|$ and k are the number of vertices and subset size, respectively. At each iteration, line 6 is successful if κ is within the current ranking bounds (q is the upper limit of this bound), and the R_{k-m+1} element will have been determined. It has been proven that this approach requires $\mathcal{O}(|V|)$ time complexity and $\mathcal{O}(|V|k)$ space complexity [23].

Algorithm 1. The unranking algorithm.

Require: $0 \le k < |V|$, $0 \le \kappa < \binom{|V|}{k}$
1: $t = |V| - k + 1$
2: $m = k$
3: R = empty list of size k
4: $q = P_{t,m} + P_{t+1,m-1} - \kappa - 1$
5: **repeat**
6: **if** $P_{t,m} \le q$ **then**
7: $R_{k-m+1} = |V| - t - m + 2$
8: $q = q - P_{t,m}$
9: $m = m - 1$
10; **else**
11: $t = t + 1$
12: **end if**
13: **until** $m = 0$
14: **return** R

It is important to note the existence of unranking algorithms having lower time and space complexity than the one utilized in this work (see [23,18] for some examples). However, in this work we did not require more advanced methods and using double-valued integer sizes allows for storage of maximum κ value (i.e. number of combinations) of approximately 1.7977e+308. If a larger number of combinations is required one can utilize cryptographic or other libraries capable of handling essentially arbitrarily large numbers.

3.1.2. Binary representation

Given the unranking approach defined above, we now reformulate the search problem to find the κ^* that corresponds to the index of the optimal solution to (9). To motivate the decision of a binary problem representation, let us consider the lexicographically ordered list L of combinations, as above, where $L(\kappa)$ is the κth element. Then due to the lexicographical ordering, $\{L(\kappa-1), L(\kappa), L(\kappa+1)\}$ will have many elements in common. Thus, there exists sufficient information amongst adjacent elements to perform a search on κ. A binary representation is attractive because it allows for both small adjustments in the corresponding decimal value (through flipping right-side bits) as well as large jumps in the search space (by left-side bit flips), while preserving the solution structure. That is, both local and global search are possible, easily implemented and efficient operations.

With respect to critical node detection: CNDP will normally require the number of critical nodes to be significantly less than the number of vertices in the graph; i.e. $k \ll |V|$. Therefore, each k-subset will encode the vertices to be removed from graph G. That is, a solution represents the graph $G(V \backslash S, W)$ for $|S| \le k$. Each solution is composed of a bit string $Q = \{0,1\}^m$, where

$$m = \left\lceil log_2 \binom{n}{k} \right\rceil \tag{11}$$

It is likely that more integer values can be represented with this string than is required. This will occur when

$$\max_{\kappa} < \binom{n}{k} \tag{12}$$

Instead of computational overhead to check the non-existence of values greater than $\max_\kappa + 1$, we simply perform a modulo operation where ø returns the integer representation of the binary argument,

$$\kappa = \phi(Q) \bmod \binom{n}{k} \tag{13}$$

Consequently, each binary solution will be converted to an integer κ that corresponds to some index

$$0 \le \kappa \le \binom{n}{k} - 1 \tag{14}$$

It must be noted that the modulo operation will potentially introduce some bias towards lower values of κ.

This binary representation also has the property of requiring less memory to store a solution since we require only sufficient bits to store a single number. Conversely, if utilizing a integer-based representation where b bits are used to encode an integer, then bk total bits are required to store k vertices. So, if

$$\left\lceil \log_2 \binom{|V|}{k} \right\rceil < bk \tag{15}$$

holds, then the binary representation is more efficient. For instance, using a 32-bit integer representation with $k = 100$ and $|V| = 1000$ we require $100 * 32 = 3200$ bits using the integer-based representation, but only 465 bits for the binary-based version.

3.1.3. From binary solution to evaluation

Decoding a solution representation for subsequent evaluation is accomplished by the procedure outlined in Algorithm 2, and highlighted by the following order of operations:

$$\underbrace{Q = \{0,1\}^m}_{\text{Representation}} \longrightarrow \underbrace{\phi(Q) \mapsto \mathbb{Z}^*}_{\text{Convert to index}} \longrightarrow \underbrace{\psi}_{\text{Unranking: generate } S} \longrightarrow \underbrace{\mathbb{Z}^*}_{\text{Evaluate } G(V \backslash S)}$$

In line 1 of Algorithm 2, Q is converted from binary to integer value and represents the index κ into the lexicographically ordered list of combinations. Using Algorithm 1 in line 2, the κ-combination is generated and stored as set $S \subseteq V$. The components of the graph $G(V \backslash S)$ are subsequently determined in line 3 and the corresponding solution is evaluated in line 4 using Eq. (9).

[1] More sophisticated methods exist that circumvent the need to store Pascal's Triangle. The method used here was chosen for simplicity. The choice of unranking method has no effect on the outcome of the proposed simulated annealing or population-based incremental learning approaches.

Algorithm 2. Decoding and evaluating a binary representation.

Require: $Q = \{0,1\}^m$, the binary representation
1: $\kappa = \phi(Q)$
2: $S = unrank(|V|,k,\kappa)$
3: $\mathcal{C} = findComponents(G \backslash S)$
4: **return** $\frac{1}{2}\sum_{c \in \mathcal{C}} \sigma_c(\sigma_c - 1)$

3.2. Population-based incremental learning

Population-based incremental learning (PBIL) is an estimation of distribution algorithm [26] that combines elements from evolutionary computation and reinforcement learning [27]. At each iteration a number of samples is generated based on the current belief about the state of each variable in the solution where the global best found solution is used to update the belief of each variable. Values in the belief vector can be probabilistically mutated in order to promote exploration of the search space.

A $d \times c$-dimensional probability matrix $\mathbf{M} \coloneqq (m_{i,j})_{d \times c}$ is defined and corresponds to a probability distribution over possible values for each element (d is the problem dimensionality, each having c variables). Learning in PBIL consists of utilizing the current $\mathbf{M}$ to generate a set H_1 of k samples. These samples are evaluated according to the objective function for the given problem and the "best" sample $\mathbf{B}^* \coloneqq (b_{i,j})_{d \times c} \in \{0,1\}$ is maintained. Then, $\mathbf{M}$ is updated by increasing the probability of generating solutions similar to $\mathbf{B}^*$ according to

$$\mathbf{M}_t = (1-\alpha)\mathbf{M}_{t-1} + \alpha\mathbf{B}^* \tag{16}$$

where $0 < \alpha < 1$ represents the learning rate parameter and subscript $t \geq 1$ corresponds to the current iteration. Without prior information $(m_{i,j}) = 0.5$. Values in $\mathbf{M}$ can be perturbed once per iteration by adding or removing a small random value to a random subset of the values in $\mathbf{M}$.

The pseudocode for PBIL is presented in Algorithm 3. It assumes constants to control the maximum number of iterations and samples, ω and k, respectively. Additionally, in line 13 parameter $0 < \gamma < 1$ controls the amount that any $m_{i,j}$ can be perturbed; according to constant $0 < \beta < 1$ in line 12.

Algorithm 3. Population-based incremental learning.

1: {Initialize probabilities}
2: $\mathbf{M}_0 \coloneqq (m_{i,j}) = 0.5$
3: **for** $t = 1$ to ω **do**
4: {Generate samples}
5: $H_1 = \text{generate\_samples}(k, \mathbf{M}_{t-1})$
6: {Find best sample}
7: $\mathbf{B}^* = \text{select\_best}(\{\mathbf{B}^*\} \cup H_1)$
8: Update $\mathbf{M}$
9: $\mathbf{M}_t = (1-\alpha)\mathbf{M}_{t-1} + \alpha\mathbf{B}^*$
10: {Mutate probability vector}
11: **for** $i = 0 \ldots d$ and $j = 0 \ldots c$ **do**
12: **if** $\mathcal{U}(0,1) < \beta$ **then**
13: $m_{i,j} = (1-\gamma)m_{i,j} + \gamma \cdot random(0 \text{ or } 1)$
14: **end if**
15: **end for**
16: **end for**

In line 2, $\mathbf{M} \coloneqq (m_{i,j}) = 0.5$ to reflect the lack of a priori information about the probability distribution of each variable. k samples are generated using the using the current probability matrix in line 5 and the best sample is retained in line 7. Matrix $\mathbf{M}$ is updated in line 9 using $\mathbf{B}^*$ to guide the direction of probability update. Finally, lines 11–15 probabilistically perform the mutation rule on elements of $\mathbf{M}$. Parameter settings used are $\alpha = 0.05$, $\beta = 0.25$, $\gamma = 0.1$; determined by limited experimentation.

It has been shown in [28–30] that for a given discrete search space PBIL will converge to a local optima. Versions of PBIL for continuous spaces have also been explored [31–34], although only the discrete binary case is considered in this work.

3.3. Simulated annealing

Simulated annealing [35] is a popular Monte Carlo optimization algorithm used for discrete and continuous optimization. Inspiration for this approach was taken from the cooling/annealing properties of metals whereby a metal is heated and cooled into a low-energy stable state. The version of the algorithm used here is presented in Algorithm 4.

The function *findBest*() in line 6 will select the solution $\mathbf{H}$ having the best evaluation from those created by the *generateNeighbors*($\mathbf{L}$,k) function, which generates $k > 0$ neighbors of the local best solution $\mathbf{L}$. A neighbor is generated according to a heuristic that selects a subset of solution features and perturbs them slightly. We tested three different methods in this work, described in Section 5. At each iteration a number of neighbors are generated, and the global best solution found thus far is tracked in line 8 and the local solution is updated according to lines 12–15. Temperature update is performed in line 17. There exist a variety of cooling approaches, each decreasing the temperature according to a different schedule [36]. In this paper we use

$$T_t = \alpha T_{t-1} \tag{17}$$

Algorithm 4. Simulated annealing algorithm.

Require: $\omega > 0, T_0 > 0, k > 0$
1: $\mathbf{B}^* = \text{random solution}$
2: $\mathbf{L} = \mathbf{B}^*$
3: $T = T_0$
4: **for** $t = 1$ to ω **do**
5: {Generate k neighbors}
6: $\mathbf{H} = findBest(generateNeighbors(\mathbf{L},k))$
7: {Maintain global best solution}
8: **if** $(eval(\mathbf{H}) < eval(\mathbf{B}^*)$ **then**
9: $\mathbf{B}^* = \mathbf{H}$
10: **end if**
11: {Update local best, or accept with probability}
12: $\Delta = eval(\mathbf{H}) - eval(\mathbf{L})$
13: **if** $\Delta < 0$ OR $e^{\Delta/T} > random(0,1)$ **then**
14: $\mathbf{L} = \mathbf{H}$
15: **end if**
16: {Update temperature}
17: $T = updateTemperature(T, T_0, t)$
18: **end for**

4. Benchmark problem instances

In order to evaluate the quality of the proposed approach, we propose 16 benchmark optimization instances based on Erdos–Renyi, Barbasi–Albert, Watts–Strogatz and Forest Fire algorithms. Other graph generation algorithms could be used for future

instances, for example [37–41]. These data sets can be downloaded at http://individual.utoronto.ca/mventresca/cnd.html, or obtained via email from the author.

4.1. Edros–Renyi model

The general procedure for creating a random graph of this type is to start with V vertices and $|E|=0$. Then, for every pair of vertices $i,j \in V$ an edge is added between them with probability p. This process defines a set of graphs $G_{V,p}$ having the same parameters V,p.

The degree distribution describes the probability of a vertex having degree k. For the Erdos–Renyi model [12] the Poisson distribution models this accurately

$$p_k = \binom{|V|}{k} p^k (1-p)^{|V|-k} \approx \frac{z^k e^{-z}}{k!} \tag{18}$$

where $z=p(|V|-1)$.

This random graph model has been of critical importance for the development of complex networks, but their structure has not been found to accurately model real world systems very well. The main difference lies in the Poisson degree distribution, whereas many real world networks follow a power law or lognormal degree distribution. Moreover, there is a lack of community structure that is often observed in real networks.

Fig. 1 shows an example Erdos–Renyi random graph with 176 vertices and 200 edges. While branches exist, the bulk of the edges are concentrated near the center of the graph where little discernible structure is observed. Another observation is the existence of multiple graph components. The Erdos–Renyi model does not enforce that the resultant graph structure have a path between all vertex pairs.

The benchmark, given in Table 1, includes four instances of the Erdos–Renyi model. They have been constructed with the properties given in Table 1. The smallest problem contains 235 vertices and 349 edges, while the largest has 2343 vertices and 3499 edges, respectively. Also given for each problem size is a set of three k-critical node values to search for.

Fig. 1. An Erdos–Renyi random graph with 235 vertices and 349 edges.

Table 1
The six Erdos–Renyi problems and associated k-critical nodes.

Name	Vertices	Edges	k-Critical nodes
ER235	235	349	50
ER465	465	699	80
ER940	940	1399	140
ER2343	2343	3499	200

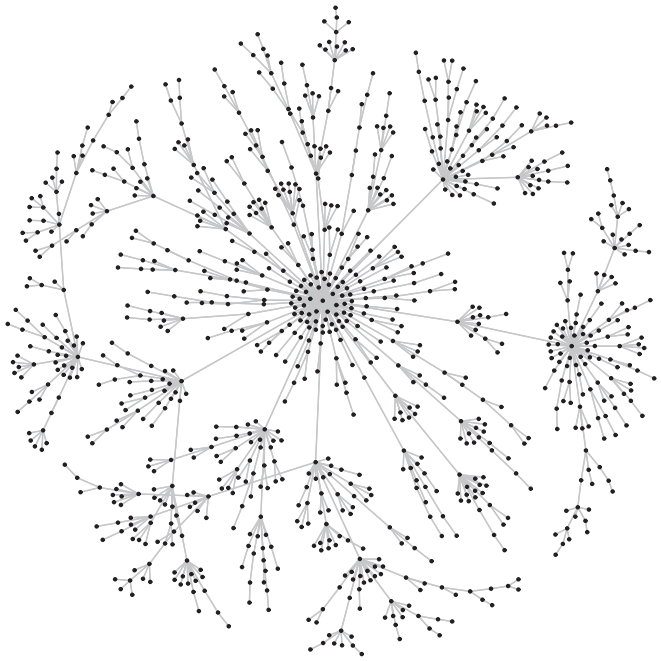

Fig. 2. An example Barabasi–Albert random graph with 1000 vertices.

4.2. Barabasi–Albert model

The Barabasi–Albert model [11] proposes that graph structure results from two processes, growth and preferential attachment. The former refers to the algorithm starting with a small set of m_0 vertices and grows the network as nodes and edges are added over time. The latter states that the probability of connecting to a node increases proportional to the current edge degree of the node; i.e. "the rich get richer".

The algorithm works as follows. The network starts with m_0 nodes and grows in stages. At each stage one vertex is added along with m edges that link the new node to m existing nodes according to the probability

$$P(e_{new,v} \in E) = \frac{deg(v)}{\sum_w deg(w)} \tag{19}$$

where $deg(\cdot)$ returns the degree of a vertex and v,w are the vertices already existing in the graph. As a result, nodes already having high degree will likely grow further. The degree distribution of this model [42] has been shown to have a power law tail with exponent 3

$$p_k \approx k^{-3} \tag{20}$$

for large k. The power law edge distribution has made this model an attractive basis for models of real networks.

Fig. 2 shows an example Barabasi–Albert graph with 1000 vertices. The "rich get richer" behavior is immediately obvious as a relatively small number of vertices act as hubs with higher connectivity. Contrasting with the Erdos–Renyi model, the entire graph forms a single component.

Table 2
The five Barabasi–Albert problems and associated k-critical nodes.

Name	Vertices	Edges	k-Critical nodes
BA500	500	499	50
BA1000	1000	999	75
BA2500	2500	2499	100
BA5000	5000	4999	150

The benchmark suite contains five Barabasi–Albert networks, shown in Table 2. The smallest network contains 500 vertices and 499 edges, whereas the largest network contains 5000 vertices and 4999 edges. The k-critical node values are also given. Due to the "rich get richer" structure of these graphs, it is expected that they will prove easiest w.r.t. the CNDP.

4.3. Watts–Strogatz small world model

Small world networks were first popularized through Milgram's degrees of separation experiments [43]. These networks have path length $L \propto \log(|V|)$. That is, the graph has a low diameter and most nodes are reachable from any other in a relatively small number of hops. These networks also exhibit a high clustering coefficient, indicating the tendency of nodes to cluster together.

The Watts–Strogatz model [13] requires as input (a) the number of vertices, (b) the number of initial neighbors of each node, b and (c) a probability p of rewiring edges between nodes. The algorithm is straightforward. Initially start with a regular lattice and continually rewire edges randomly throughout the graph with probability p. During this graph generation process, the further apart the new edge ending is from its start the more reduction in graph diameter is observed. The best reduction ratio achievable is 0.5 at any step, and so after a small number of iterations the graph diameter will have been dramatically reduced. Except when the rewiring probability is zero, the degree distribution is Poissonian.

Fig. 3 shows an example Watts–Strogatz network with 250 vertices, initial neighborhood size of 5 and rewiring probability $p=0.05$. This particular model has a clustering coefficient of ≈ 0.47 and a diameter of 6. Given the density of connections and small-world property, these networks will require removal of a significant number of vertices in order to limit the size of the largest graph component.

Table 3 shows the four benchmark instances. Networks WS250 and WS1000 were generated using an initial neighbor value of 5 and 0.05 rewiring probability. Instance WS500 used an initial neighbor value of 3 and 0.05 rewiring probability, and WS1500 used a neighbor value of 3 and rewiring of 0.03. The smallest network has 250 nodes and 1250 edges, and the largest graph has 1500 vertices and 4500 edges. The small-world property of these networks should result in a difficult optimization, or a very undesirable optimal solution (giant component is very large).

4.4. Forest Fire model

We also use the Forest Fire random graph model [14] to generate benchmark data. Like the Barabasi–Albert model, this is also a preferential attachment approach. However, this model reproduces heavy tailed degree distribution where the network diameter decreases over time. As a result, this model also follows a densification power-law whereby the network becomes dense according to a power-law rule. The name of the model is derived from the resemblance of its growth pattern to the manner in which forest fires spread.

Fig. 3. An example Watts–Strogatz small world model with 250 vertices.

Table 3
The four Watts–Strogatz problems and associated k-critical nodes.

Name	Vertices	Edges	k-Critical nodes
WS250	250	1250	70
WS500	500	1500	125
WS1000	1000	5000	200
WS1500	1500	4500	265

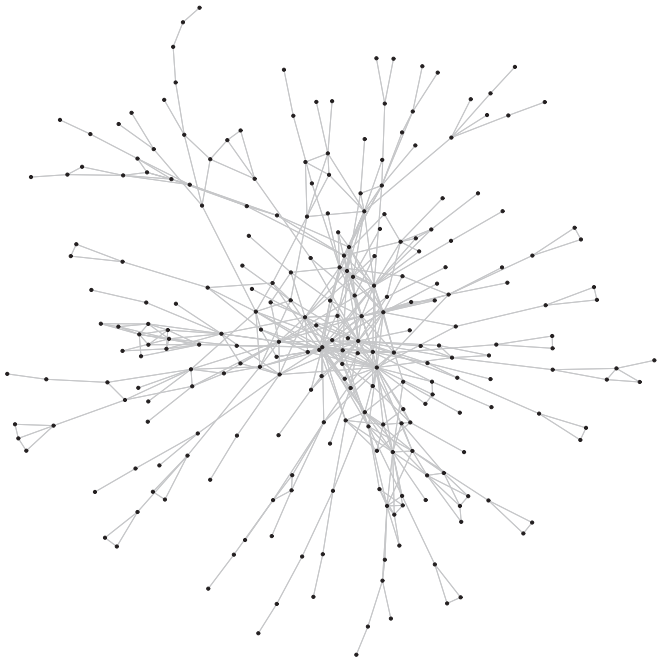

Fig. 4. An example Forest Fire random graph model with forward probability $p=0.25$ and $r=0.2$ backward probability, respectively.

The algorithm to generate Forest Fire networks is also straightforward. The network starts with at random vertex $v \in V$ and uniformly selects some other vertex $w \in V$ and forms a connection

(v,w). Then, select x out-links and y in-links of w, where x and y are the binomially distributed with means

$$\frac{p}{1-p} \quad \text{and} \quad \frac{rp}{1-rp} \tag{21}$$

respectively, and $0 < p$, $r < 1$ are probabilities. Create links from v to each of the $x+y$ edges and recursively repeat the process from the $x+y$ vertices.

An example Forest Fire graph with forward probability $p=0.25$ and backward probability $p=0.20$ is given in Fig. 4. This structure seems to be a mix of the Erdos–Renyi and Barabasi–Albert models. This model does have dense connection regions as well as long branches.

Table 4 gives the four Forest Fire problem instances in the benchmark. All networks were created with forward probability $p=0.25$ and backward probability 0.20. The smallest network is made of only 250 vertices and 400 edges, and the largest is constructed of 2500 vertices and 4046 edges. The Forest Fire model should be harder for critical node detection than the Barabasi–Albert model as it contains more edges for the same sized graph (with the parameters used here).

Table 4
The five Forest-Fire problems and associated k-critical nodes.

Name	Vertices	Edges	k-Critical nodes
FF250	250	400	13
FF500	500	792	25
FF1000	1000	1633	50
FF2500	2500	4046	125

Table 5
The average runtime (in seconds) of SA and PBIL for the 16 benchmark functions.

Problem	SA	PBIL	Problem	SA	PBIL
ER235	38	84	WS250	70	135
ER466	110	183	WS500	173	263
ER941	361	469	WS1000	548	676
ER2344	1931	2171	WS1500	1816	2064
BA500	66	126	FF250	37	88
BA1000	172	264	FF500	156	233
BA2500	840	1178	FF1000	410	509
BA5000	3154	3515	FF2000	1723	1961

5. Experimental results

In this section we explore the quality of the proposed algorithms on the 16 aforementioned problems. Both SA and PBIL will require a perturbation operator during the optimization. We experimented with three operators, and selected the best performing over all the 16 instances, without tuning parameters to work best for individual problems. The first subsection details these operators, but we do not show experimentation for them. Subsequent sections will utilize the operator that performed most admirably. In all cases, the algorithms were run for 30 trials with the number of samples per iteration, and run lengths determined by the number of vertices in the given graph, according to:

$$\{\text{samples, iterations}\}(|V|) = \begin{cases} \{300, 2500\} & \text{if } |V| \leq 500 \\ \{400, 4000\} & \text{if } |V| \in (500, 1000] \\ \{500, 6000\} & \text{if } |V| \in (1000, 2500] \\ \{600, 7500\} & \text{if } |V| \in (2500, 5000] \end{cases} \tag{22}$$

Other parameter settings for the algorithms are as follows: SA initial temperature=# iterations $*|V|$ and temperature, T, update was performed every other iteration using $T_{i+1} = 0.985 * T_i$. PBIL parameters were experimentally determined to be $\alpha = 0.03$, $\beta = 0.05$ and $\gamma = 0.05$. All experiments were conducted on an Intel i7-2600 K (quad-core, eight threads 3.4 GHz) machine with 8 GB RAM, with the algorithms being programmed using version 1.6 of Sun Java.

5.1. Sample generation and perturbation operator

We considered three types of perturbation/update operator, for SA and PBIL, respectively. The first method considers all dimensions of the solution representation and performs a perturbation with fixed probability p, which was tested to vary by 0.005 increments over the range [0.005, 0.10]. This type of operator allows for maximum potential of search-space exploration at any given iteration. However, the number of dimensions could lead to longer convergence times, or difficulty escaping poor local optima.

Table 6
Summary results (mean, standard deviation, minimum and maximum) for SA and PBIL.

Problem	SA				PBIL			
	μ	σ	min	max	μ	σ	min	max
ER235	8697.6	437.6	7700	9501	7849	407.5	6700	8342
ER466	51,830	1109.1	48,627	53,353	49,892.1	1319	44,255	51,434
ER941	242,350.2	2814.8	234,479	246,133	238,769	2174.2	229,576	241,281
ER2344	2,035,371.9	7907.6	2,011,122	2,047,327	2,024,226.8	6365.2	2,009,132	2,033,234
BA500	1401.4	416.7	997	2878	1124	85	892	1272
BA1000	6527.5	1545	3770	11,821	5080.9	1175.5	3057	7748
BA2500	38,080.7	6344.9	31,171	55,336	33,188.5	2478.5	28,044	38,402
BA5000	289,805.3	173,126.9	170,998	900,448	187,713.6	23,307.4	146,753	244,057
WS250	15,114.1	370	14,251	15,579	14,776	337.4	13,786	15,235
WS500	60,425.4	1749.6	54,201	62,203	58,394.7	1316.1	53,779	60,277
WS1000	316,764.4	1790.9	311,700	318,801	314,447.3	1816.3	308,596	317,209
WS1500	723,795.4	3642.8	717,369	730,561	715,519	4489.3	703,241	722,162
FF250	4745.6	1493.1	1841	7645	2906.5	1024.8	1386	5905
FF500	4745.3	2520.9	2397	12,598	2532.8	400.2	1904	3480
FF1000	127,211.2	15,146.3	92,800	157,248	102,720.4	21,026.4	59,594	135,670
FF2000	751,493.6	129,468.6	387,248	906,680	543,573.7	135,291.5	256,905	765,222

The second operator focuses on a contiguous subset $\mathcal{W} \subseteq \{1,..,m\}$, i.e. an interval of indices in the representation, at each iteration. This tactic could allow for exploitation of the best known solution structure while exploring the best arrangement of a subset of the elements. The number of elements in $\mathcal{W}$ is determined by

$$|\mathcal{W}| = \lfloor m^{-2.5} + \log_2(m) \rfloor \quad (23)$$

where m is according to Eq. (11). The SA algorithm uses a window size twice this size. During optimization this window is moved at each iteration from the lowermost bit towards the uppermost bit, repeating once the uppermost bit is reached. The final operator is similar to the sliding window approach. Instead of the window, a random subset of size $|\mathcal{W}|$ elements is selected. This approach is intended as a less biased version of the windowing technique. PBIL probabilities must always exist in the range [0.5, 0.95].

Generating solutions at each iteration has a significant impact on the solution quality as well. To fully utilize the search ability, we employ a strategy that uses the best known solution as a template background and only sample variations within $\mathcal{W}$. Experimentation led to the use of the sliding window technique for the sampling phase of SA, with a probability $2/|\mathcal{W}|$ on each

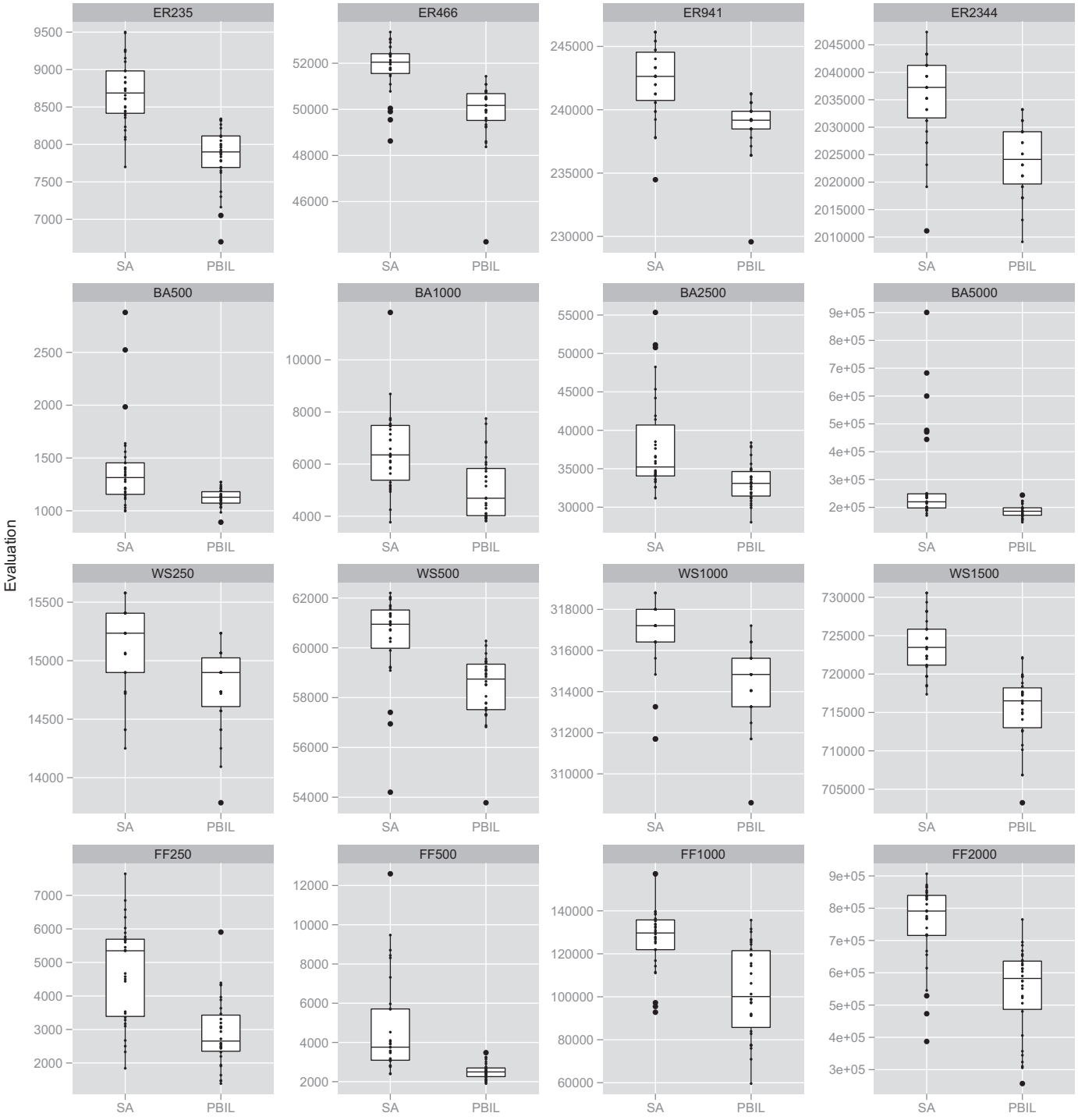

Fig. 5. Box and whisker plots of results for the 16 benchmark problems.

element. That is, we expect two perturbations per sample. PBIL also utilized this windowed version, but the perturbation operator utilized the randomized window approach.

5.2. Running time

In this subsection we perform a comparison of the aforementioned SA and PBIL algorithms using the 16 benchmark graphs. The average runtime (in seconds) for each algorithm is given in Table 5. The overhead of PBIL's probability update step is evident, but as problem size increases the difference from SA is decreased. Given the size of the networks under consideration, the number of iterations and the number of samples per iteration, these are very promising results. For instance, solving the WS1500 problem will involve performing 6000 iterations each with 500 samples, thus 3,000,000 candidate solutions are generated. More traditional approaches have been limited to 1000s to determine a bounded solution to 50 or 100 node problems [10], or required many thousands of seconds to determine an optimal [1] for problems with less than 150 vertices. Moreover, the best existing heuristic [1] is also much less efficient than either the SA or PBIL approach presented here.

5.3. Comparing results

Unfortunately, comparison of our results to pre-existing approaches was not possible due to either limited RAM or exceptionally high computational demands, but we are hopeful that such a comparison will become feasible in the near future. A summary of our experimental results are presented in Table 6. We show the mean (μ), standard deviation (σ), minimum and maximum obtained result for each algorithm, respectively. Corresponding box-and-whisker for each experiment, showing the distribution of results is given in Fig. 5.

For many of the proposed problems PBIL is more likely to yield a better quality outcome, although SA can produce reasonable results as well. Results obtained with PBIL tend to be more reliable, as indicated by the σ values, than SA and the range of minimum/maximum values. The standard deviation also provides some minimal insight into the range of problem difficulty of the 16 benchmark instances. Problems with highly variable results may be more difficult to solve, whereas less variable results may indicate easier problems (or those with a difficult to discover optima). A full analysis of problem difficulty is beyond the scope of this paper, however, would form an interesting direction for future work.

Table 7
Comparing the effect size of the difference between SA and PBIL results.

Problem	Cohen-d	Interp.	Favors
ER235	2.04	Large	PBIL
ER466	1.62	Large	PBIL
ER941	1.45	Large	PBIL
ER2344	1.58	Large	PBIL
BA500	0.94	Large	PBIL
BA1000	1.07	Large	PBIL
BA2500	1.03	Large	PBIL
BA5000	0.84	Large	PBIL
WS250	0.97	Large	PBIL
WS500	1.33	Large	PBIL
WS1000	1.31	Large	PBIL
WS1500	2.06	Large	PBIL
FF250	1.46	Large	PBIL
FF500	1.25	Large	PBIL
FF1000	1.36	Large	PBIL
FF2000	1.60	Large	PBIL

The practical effect of the observed difference between SA and PBIL is estimated using Cohen's d-statistic, calculated as

$$d = \frac{\overline{x}_1 - \overline{x}_2}{s} \tag{24}$$

where $\overline{x}_1$ and $\overline{x}_2$ are the means of two results with n_1 and n_2 observations in each group, respectively. The pooled standard deviation s is calculated according to

$$s = \sqrt{\frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n_1+n_2}} \tag{25}$$

where s_1 and s_2 are the group 1 and group 2 standard deviations. Cohen's d-statistic can be interpreted using the following heuristic: the levels are $d > 0.8$, $0.5 < d < 0.8$, $0.2 < d < 0.5$ and < 0.2, which correspond to degrees of differences of large, medium, small and none, respectively. The results of this analysis are presented in Table 7. For all problems the PBIL approach significantly outperforms SA. A Kolmogorov–Smirnov test was also performed to determine whether the values could have been drawn from the same distribution. It was confirmed that this is

Table 8
Assessing the quality of the proposed algorithms versus a random sampling approach. Values for the μ and min columns are given as percentages.

Problem	RANDOM	SA		PBIL	
	min	μ	min	μ	min
ER235	8106	−7.3	5	3.2	17.3
ER466	49,391	−4.9	1.5	−1	10.4
ER941	238,498	−1.6	1.7	−0.1	3.7
ER2344	2,021,146	−0.7	0.5	−0.2	0.6
BA500	1627	13.9	38.7	30.9	45.2
BA1000	7051	7.4	46.5	27.9	55.7
BA2500	57,958	34.3	46.2	42.7	51.6
BA5000	353,766	18.1	51.7	46.9	58.5
WS250	15,066	−0.3	5.4	1.9	8.5
WS500	59,118	−2.2	8.3	1.2	9
WS1000	312,481	−1.4	0.2	−0.6	1.2
WS1500	713,747	−1.4	−0.5	−0.2	1.5
FF250	3463	−37	46.8	16.1	60
FF500	3732	−27.2	35.8	32.1	49
FF1000	125,680	−1.2	26.2	18.3	52.6
FF2000	609,883	−23.2	36.5	10.9	57.9

Table 9
Summary results of the largest component achieved by the SA and PBIL algorithms.

Problem	SA				PBIL			
	μ	σ	min	max	μ	σ	min	max
ER250	131.4	3.6	124	138	124.3	4.1	110	129
ER500	322.1	3.5	312	327	315.9	4.6	296	321
ER1000	696.6	4.1	685	702	691.4	3.2	678	695
ER2500	2018.1	3.9	2006	2024	2012.5	3.2	2005	2017
BA500	22.9	7.6	14	46	18.4	2.4	15	23
BA1000	56.1	11.4	29	86	46.7	12.7	27	73
BA2500	110.6	29.3	77	171	90.8	15.4	68	128
BA5000	386.9	207.1	188	992	265	32.6	216	344
WS250	174.3	2.2	169	177	172.2	2.1	166	175
WS500	347.3	5.3	329	353	340.9	4.3	325	347
WS1000	796.4	2.3	790	799	793.5	2.3	786	797
WS1500	1203.3	3	1198	1209	1196.4	3.8	1186	1202
FF250	92	19.9	39	123	65.1	19.1	33	107
FF500	62.7	32.1	30	154	35.5	6.7	26	52
FF1000	497.9	34.9	415	559	438.6	57.4	311	519
FF2000	1210.9	126.1	790	1344	990.6	202.3	469	1234

highly unlikely as all p-values for this hypothesis were negligible (< 0.00001).

Given the lack of comparative algorithms, we will gauge the quality of the SA and PBIL algorithms versus the best result obtained by random sampling. While this is not a perfect measure, it provides insight into both the quality of the proposed methods, as well as the difficulty of the problems themselves. For each instance we generate the same number of random samples as one would have generated for all runs of SA or PBIL. For example, WS1500 generates 3,000,000 samples per run and thus we generate 30 times this number (recall, 30 trials are conducted for SA and PBIL). We retain the minimum, versus the mean, of the random sampling in order to be as critical as possible on the proposed algorithms.

Table 8 presents the results of our assessment. For each problem the minimum value found by random sampling is given, as well as the percentage difference of the mean and best results found by SA and PBIL (as given in Table 6). This difference is computed as a percentage according to

$$100 \cdot \frac{RANDOM - ALG}{RANDOM} \tag{26}$$

where ALG is substituted for the mean or minimum result obtained by SA or PBIL. Negative values indicate the random sampler is more desirable. The larger this percentage the more desirable the outcome for the algorithm is.

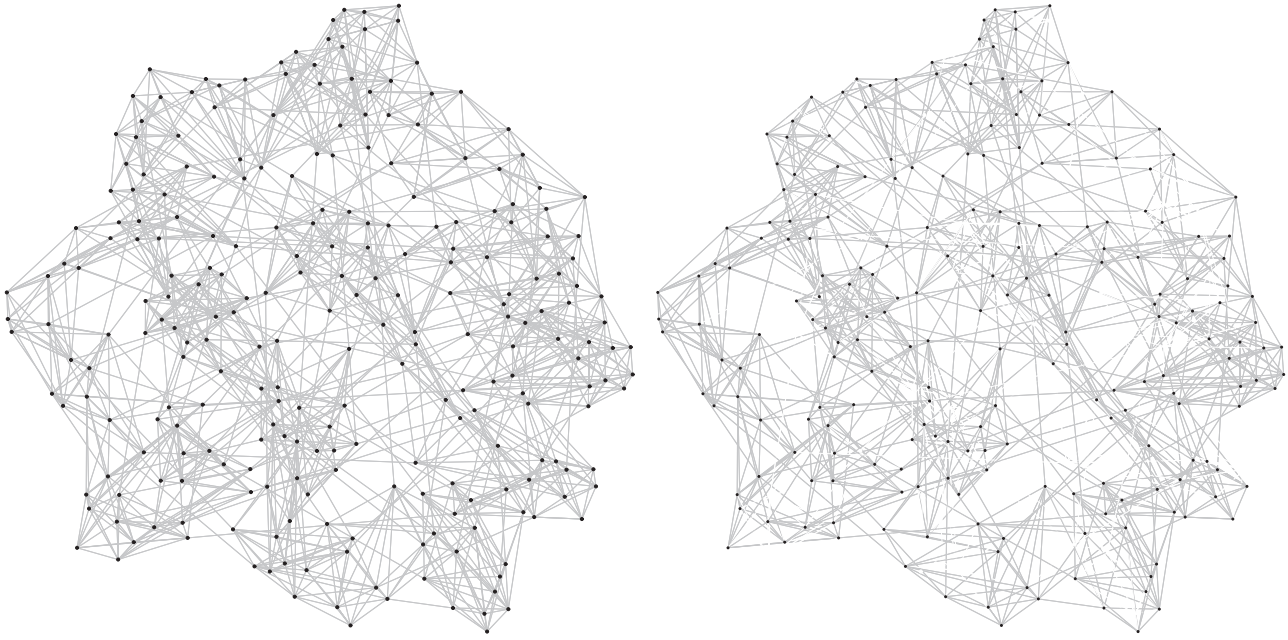

Fig. 6. Watts–Strogatz model (left) of 250 vertices, and after removal of 70 vertices (right). The high clustering coefficient and small diameter relate to very dense graphs that are difficult to compartmentalize.

Fig. 7. Erdos–Reni random graph (left) with 235 vertices, and after removal of 50 vertices (right). The network has been segmented after removing critical nodes, but a very large component still remains.

The mean SA outcome turns out to be less desirable than the best RANDOM solution, for all but the Barabasi–Albert instances. These graphs are the simplest to solve, and as indicated above polynomial time algorithms exist to solve them optimally. Most instances the mean PBIL is more desirable than RANDOM, and cases where this is not true the difference is negligible. For both SA and PBIL the Forest Fire and Barabasi–Albert instances seem easiest to solve, whereas the Erdos–Renyi and Watts–Strogatz models seem significantly more difficult. As indicated previously, these graphs have characteristics such as small diameter or higher clustering coefficients that will likely make them more challenging to solve.

When comparing the best results obtained by each algorithm to the random approach we see that the pattern of problem difficulty noticed above remains. Also the results are, as expected, more desirable than one would find using the purely random sampler.

5.4. Sample solution graphs

In this section sample graphs are shown to illustrate the type of solution represented by the above optimizations. This is done by presenting the graph before and after removal of the designated number of vertices. For space limitations, as well as visualization limitations we restrict the plots to networks having 250, 500 or 1000 vertices. Firstly, we present summary results concerning the size of the largest component in the graph.

Identification of the largest component is important in a number of practical applications. While not the explicit goal of the CNDP formulation, it is desirable to make the size of this

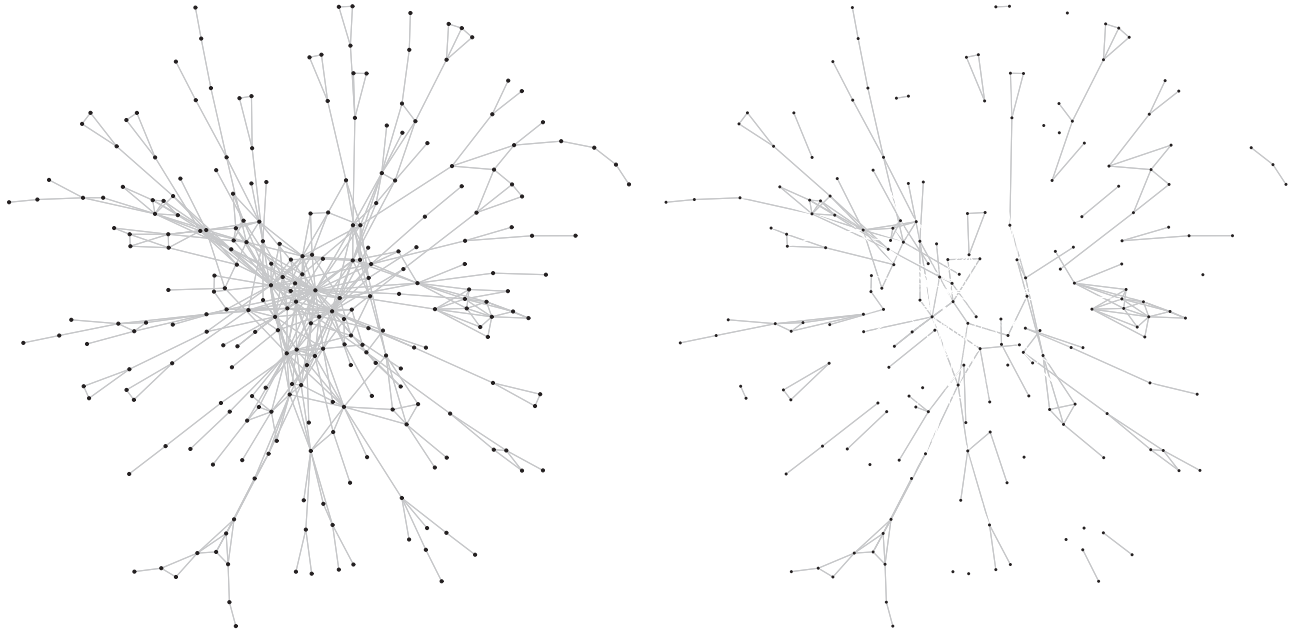

Fig. 8. Forest Fire network model of 250 vertices (left). Removal of 70 critical nodes leaves the network somewhat segmented (right).

Fig. 9. Barabasi–Albert preferential attachment model of 1000 vertices (left). After removing 75 critical nodes the network is significantly fragmented.

component as small as possible. Table 9 presents summary results for the largest component size for each problem instance and optimization algorithm. PBIL tends to yield the smaller component, which is expected from the above results. However, it is quite interesting to note the relatively small differences between the component size of these two approaches. Specifically, that even a largest component size difference of 5–10 vertices (2–4% of a 250 vertex graph) can have such a large impact on the network as a whole.

Figs. 6–9 show typical results from optimization with PBIL. The WS250 problem before and after removal of 70 critical nodes is less dense, however, the network remains highly connected with the largest component size of 172 vertices. Significant reduction in this value will require a very large number of critical nodes, but nonetheless this is a very challenging problem to optimize. ER250 is also plotted (Fig. 7) before and after removal of 50 vertices. The network is characteristically random with respect to the connection structure and has a largest component size of 119, approximately half of the original network size. Fig. 8 shows the Forest Fire model with 250 vertices. Fifty critical nodes are identified and removed, with the remaining largest component having only 39 vertices. Finally, in Fig. 9 a Barabasi–Albert model with 1000 vertices and 75 critical nodes is shown. The high degree of segmentation is expected as these models are tree-like and shown to be simple to solve [9].

6. Conclusions and possible future work

This paper proposed simulated annealing and population-based incremental learning approaches for attaining solutions to the critical node detection problem. Neither method had been previously applied in this context previously. A sliding window-based perturbation operator was designed, which employed an efficient unranking-based problem representation and shown to yield quality results. An important additional benefit of this representation is the alleviation of any repair mechanism.

In order to assess the quality of the algorithms 16 benchmark problem instances were also proposed. These problems are created from Erdos–Renyi, Barabasi–Albert, Watts–Strogatz and Forest Fire complex network models and represent varying degrees of difficulty for an optimization algorithm. Additionally, their relatively large size (compared to previous problem instances of less than 250 vertices) make efficient computation necessary. Our statistical analyses reveals that the PBIL-based algorithm is superior to SA, for these problems.

Future works include creating larger benchmark problems, on the order of hundreds of thousands or millions of vertices. The representation presented here could be used for these extremely large problems as our representation scales logarithmically with respect to the number of vertices. That is, we use a compressed representation and this would therefore have a smaller search space at large networks than otherwise would be required. However, further theoretical and practical study of the solution representation is required. Other directions for future endeavors also include using other network models and heuristics.

Acknowledgements

The author would like to thank the anonymous reviewers for their constructive input.

References

[1] Arulselvan A, Commander CW, Elefteriadou L, Pardalos PM. Detecting critical nodes in sparse graphs. Computers and Operations Research 2009;36(7):2193–200.
[2] Boginski V, Commander C. Identifying critical nodes in protein–protein interaction networks. In: Clustering challenges in biological networks; 2009. p. 153–66.
[3] Nian F, Wang X. Efficient immunization strategies on complex networks. Journal of Theoretical Biology 2010;264(1):77–83.
[4] Jorgic M, Stojmenovic I, Hauspie M, Simplot-Ryl D. Localized algorithms for detection of critical nodes and links for connectivity in ad hoc networks. In: Third annual IFIP mediterranean ad hoc networking workshop, Med-Hoc-Net; 2004. p. 360–71.
[5] Arulselvan A, Commander C, Pardalos P, Shylo O. Managing network risk via critical node identification. In: Gulpinar N, Rustem B, editors. Risk management in telecommunication networks. Springer; 2009.
[6] Fan N, Pardalos P. Robust optimization of graph partitioning and critical node detection in analyzing networks. In: Fourth international conference on combinatorial optimization and applications; 2010. p. 170–83.
[7] Arulselvan C, Commander A, Shylo O, Pardalos P. Cardinality-constrained critical node detection problem. In: Performance models and risk management in communications systems; 2011. p. 79–91.
[8] Di Summa M, Grosso A, Locatelli M. Complexity of the critical node problem over trees. Computers and Operations Research 2011;38(12):1766–74.
[9] Addis B, Di Summa M, Grosso A. Removing critical nodes from a graph: complexity results and polynomial algorithms for the case of bounded treewidth. Discrete Applied Mathematics, submitted for publications. 〈http://www.optimization-online.org/DB_HTML/2011/07/3112.html〉.
[10] Di Summa M, Grosso A, Locatelli M. Branch and cut algorithms for detecting critical nodes in undirected graphs. Computational Optimization and Applications, submitted for publication. 〈http://www.springerlink.com/content/k7h6518235573704/〉.
[11] Barabasi A, Reka A. Emergence of scaling in random networks. Science 1999;286:509–12.
[12] Erdos P, Renyi A. On random graphs. I. Publicationes Mathematicae 1959;6:290–329.
[13] Watts D, Strogatz S. Collective dynamics of 'small-world' networks. Nature 1998;393:400–42.
[14] Leskovec J, Kleinberg J, Faloutsos C. Graphs over time: densification laws. In: Proceeding of the 11th ACM international conference on knowledge discovery in data mining; 2005. p. 177–87.
[15] Cormen T, Leiserson C, Rivest R, Stein C. Introduction to algorithms. third ed.MIT Press; 2009.
[16] Carey MR, Johnson DS. Computers and intractability. W.H. Freeman and Company; 1979.
[17] Kreher D, Stinson D. Combinatorial algorithms: generation enumeration and search. CRC Press; 1998.
[18] Nayak A, Stojmenovic I, editors. Handbook of applied algorithms: solving scientific, engineering and practical problems. Wiley-Blackwell; 2008.
[19] Kokosinski Z. A parallel dynamic programming algorithm for unranking tary trees. In: Wyrzykowski R, Dongarra J, Paprzycki M, Wasniewski J, editors. Parallel processing and applied mathematics. Lecture notes in computer science, vol. 3019. Springer; 2004. p. 255–60.
[20] Flajolet P, Zimmerman P, Van Cutsem B. A calculus for the random generation of combinatorial structures. Theoretical Computer Science 1994;132(1):1–35.
[21] Myrvold W, Ruskey F. Ranking and unranking permutations in linear time. Information Processing Letters 2001;79(6):281–4.
[22] Martinez C, Molinero X. An experimental study of unranking algorithms. In: Ribeiro CC, Martins SL, editors. Experimental and efficient algorithms. Lecture notes in computer science, vol. 3059; 2004. p. 326–40.
[23] Kokosinski A. Algorithms for unranking combinations and other related choice functions. Technical Report, Department of Computer Software, The University of Aizu; 1995.
[24] Critani F, DallAglio M, Biase GD. Ranking and unranking permutations with applications. Innovation in Mathematics 1997;15(3):99–106.
[25] Skiena S. The algorithm design manual. Springer; 2008.
[26] Larranaga P, Lozano J, editors. Estimation of distribution algorithms: a new tool for evolutionary computation. Springer; 2001.
[27] Baluja S. Population based incremental learning—a method for integrating genetic search based function optimization and competitive learning. Technical Report, Carnegie Mellon University, CMU-CS-94-163; 1994.
[28] Rastegar R, Hariri A, Mazoochi M. A convergence proof for the population based incremental learning algorithm. In: Seventeenth IEEE international conference on tools with artificial intelligence; 2005. p. 387–91.
[29] Hohfeld M, Rudolph G. Towards a theory of population-based incremental learning. In: Proceedings of the fourth IEEE Conference on evolutionary computation; 1997. p. 1–5.
[30] Rastegar R, Hariri A. The population-based incremental learning algorithm converges to local optima. Neurocomputing 2006;69(13–15):1772–5.
[31] Rudlof S, Koppen M. Stochastic hill climbing with learning by vectors of normal distributions. In: First on-line workshop on soft computing; 1996. p. 60–70.
[32] Sebag M, Ducoulombier A. Extending population-based incremental learning to continuous search spaces. In: PPSN V: Proceedings of the fifth international conference on parallel problem solving from nature. Springer; 1998. p. 418–27.
[33] Muhlenbein H, PaaB G. From recombination of genes to the estimation of distributions. I. binary parameters. In: Voigt H-M, Ebeling W, Rechenberg I, Schwefel H-P, editors. Parallel problem solving from nature IV. Lecture notes in computer science, vol. 1141. Springer; 1996. p. 178–87.

[34] Gallagher M, Frean M. Population-based continuous optimization, probabilistic modelling and mean shift. Evolutionary Computation 2005;13(1):29–42.
[35] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220(4598):671–80.
[36] Nourani Y, Andresen B. A comparison of simulated annealing cooling strategies. Journal of Physics A: Mathematical and General 1998:8373–85.
[37] Aiello W, Chung F, Lu L. A random graph model for massive graphs. In: ACM symposium on theory of computing; 2000. p. 171–80.
[38] Barabasi A-L, Jeong H, Neda Z, Ravasz E, Schubert A, Vicsek T. Evolution of the social network of scientific collaborations. Physica A 2002;311: 590–614.
[39] Dorogovtsev SN, Mendes JF. Evolution of networks: from biological nets to the internet and WWW. Oxford University Press; 2003.
[40] Chakrabarti D, Zhan Y, Faloutsos C. R-mat: a recursive model for graph mining. In: SIAM international conference on data mining; 2004. p. 442–6.
[41] Newman MEJ, Strogatz SH, Watts DJ. Random graphs with arbitrary degree distributions and their applications. Physical Review E 64(2).
[42] Dorogovtsev SN, Mendes JF, Samukhin AN. Structure of growing networks with preferential linking. Physical Review Letters 2000;85:4633–6.
[43] Milgram S. The small world problem. Psychology Today 1967;1(1):60–7.