Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

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

Analysis of an iterated local search algorithm for vertex cover in sparse random graphs[☆]

Carsten Witt[*]

DTU Informatics, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark

ARTICLE INFO

Keywords:
Randomized search heuristics
Iterated local search
Vertex cover
Random graphs
Karp–Sipser algorithm
e-phenonemon

ABSTRACT

Recently, various randomized search heuristics have been studied for the solution of the minimum vertex cover problem, in particular for sparse random instances according to the $G(n, c/n)$ model, where $c > 0$ is a constant. Methods from statistical physics suggest that the problem is easy if $c < e$. This work starts with a rigorous explanation for this claim based on the refined analysis of the Karp–Sipser algorithm by Aronson et al. (1998) [1]. Subsequently, theoretical supplements are given to experimental studies of search heuristics on random graphs. For $c < 1$, an iterated local search heuristic finds an optimal cover in polynomial time with a probability arbitrarily close to 1. This behavior relies on the absence of a giant component. As an additional insight into the randomized search, it is shown that the heuristic fails badly also on graphs consisting of a single tree component of maximum degree 3.

1. Introduction

Randomized search heuristics (RSHs) such as Evolutionary Algorithms (EAs), Simulated Annealing, Ant Colony Optimization etc. are general optimization techniques that prevail in applications where problem-specific algorithms are not available. In the last few years, substantial progress has been made in the rigorous runtime analysis of RSHs for problems from combinatorial optimization [23,11,22,18,9,8,14]. In these works, the key question is how long the heuristics take in expectation to find a solution of a prespecified quality.

Recently, the behavior of RSHs for the minimum vertex cover (VC) problem has received increasing attention [9,8]. These studies were concerned with specific instances of the problem. A mostly empirical work [19] studies an average case where the graph is drawn randomly according to the $G(n, p)$ model [3] with $p = c/n$ for constant c. In particular, that paper highlights the well-known phase transition in this model. If $c < 1$ then all connected components of the graph are of size $O(\log n)$ with high probability $1 - o(1)$ (abbreviated as *w.h.p.* hereinafter), and the problem is easy to solve by complete enumeration. Actually, by references to methods from statistical physics [12], it is claimed in [19] that VC is easy in random graphs if $c < e$. However, no rigorous argument is given.

One aim of this work is to supply theoretical justifications to the experimental analyses of RSHs for VC in sparse random graphs. We start with a rigorous proof in Section 2 that VC in sparse random graphs can be solved in polynomial time w. h. p. for $c < e$. Despite being a simple consequence of the refined analysis of the Karp–Sipser algorithm presented by Aronson et al. [1], this result does not yet seem to have been stated explicitly so far (at least in the community of theoretical computer science). Afterwards, we study the behavior of a simple RSH in our random graph model. An iterated local search algorithm called ILS that combines ingredients of blind randomized local search with a greedy component is defined in Section 3. It is

☆ An extended abstract of this work appeared in Witt (2009) [24].

* Tel.: +45 492317552469.
E-mail address: cawi@imm.dtu.dk.

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.01.010

proved to find optimal VCs in the domain $c < 1$ with probability $1-\epsilon$, $\epsilon > 0$ an arbitrary small constant. This result relies on the absence of the giant component, but it still comes unexpected since the local search is far from complete enumeration. In order to fathom the limits of the approach, it is shown that ILS fails even on trees of small degree when the tree forms a giant component. We finish with some conclusions.

2. A modified Karp–Sipser algorithm for vertex cover

The vertex cover problem (occasionally also called node cover problem) is one of the classical NP-hard problems [15]. Given an undirected graph $G = (V, E)$, the aim is to find a subset $V^* \subseteq V$ of minimum cardinality such that every edge is covered, i. e., for all $\{v, w\} \in E$ it holds $v \in V^*$ or $w \in V^*$ (or both). The hardness of the problem motivated the study of average-case models, one of which is the random graph model called $G(n, c/n)$ [3]. Here a graph on n vertices is built by inserting every possible edge independently with probability c/n, where c is a constant. This results in the expected vertex degree $c(n-1)/n$ and the expected number of edges $c(n-1)/2$. The graph is typically called *sparse* since its expected number of edges is only linear in the number of vertices.

In [19], various heuristics including EAs, SA, and branch and bound are studied for the VC problem in sparse random graphs drawn according to the $G(n, c/n)$ model. While evaluating their experiments, Pelikan et al. [19] claim that the problem is "typically" polynomial-time solvable for $c < e$, where $e = 2.718\ldots$ is the base of the natural logarithm. This phenomenon is explained by a reference to methods from statistical physics [12,2]. The underlying idea described by Bauer and Golinelli [2] is to study the application of a procedure called *leaf-removal* to the input graph: as long as the graph has at least one leaf (a vertex of degree 1), choose such a leaf uniformly and delete the leaf and its neighbor (along with incident edges) from the graph. The graph that finally remains is called the "core" by Bauer and Golinelli [2] (a notion different from the well-known (k-)core of a graph). The non-rigorous analysis using statistical mechanics reveals that the "core" is of size $O(\log n)$ provided that $c < e$. Hence, it is proposed by Bauer and Golinelli [2] to solve the VC problem by applying leaf-removal, putting cover marks on the leaves' neighbors, and finally solving the VC problem on the "core" using branch and bound as a brute-force approach.

Roughly speaking, Bauer and Golinelli [2] identify a second phase transition in the $G(n, c/n)$ model besides the well-known emergence of a giant component at $c = 1$: namely, the emergence of a giant "core" at $c = e$. They also relate the latter to the so-called e-phenomenon first observed by Karp and Sipser [16] w. r. t. the maximum matching problem and studied in more detail by Aronson et al. [1]: i. e., the leaf-removal approach can also be applied to find a large matching (a set of vertex-disjoint edges) in the graph. The famous Karp–Sipser algorithm [16] selects edges incident on leaves for the matching and reduces the graph afterwards, i. e., it performs leaf-removal in the above sense, until there are no leaves left (the subsequent behavior of Karp–Sipser is not important for this paper). If $c < e$, the "core" remaining after leaf-removal is of asymptotically negligible size w. h. p., hence the set of edges chosen by leaf-removal yields a $(1 - o(1))$-approximation of a maximum matching.

The reference to the Karp–Sipser algorithm is more or less a side remark in the paper by Bauer and Golinelli [2]. From the viewpoint of theoretical computer science, we are aiming at making the interplay of the Karp–Sipser algorithm, the "core", maximum matchings and minimum VC more explicit. In particular, we are interested in a rigorous statement regarding the "typical" $O(\log n)$ size claimed by Bauer and Golinelli [2] for the "core" graph. Fortunately, such a statement is already contained in the analysis by Aronson et al. [1]. This results in the forthcoming Theorem 2 that VC in sparse random graphs can be solved to optimality w. h. p. if $c < e$. The algorithm used is a modification of the Karp–Sipser algorithm for the VC problem, called KS-VC and described in Algorithm 1. The notion $G \setminus N$ for a set of vertices N denotes the subgraph of G induced by $V(G) \setminus N$. By $V(G)$ and $E(G)$ we denote the set of vertices and edges of G, respectively.

Algorithm 1 (*KS-VC*).

```
1. $C := \emptyset$.
2. While $E(G) \neq \emptyset$
     If $G$ has at least one leaf then
       choose a leaf $w \in V(G)$ uniformly,
       let $\{v, w\} \in E(G)$ be the unique edge incident on $w$,
       $C := C \cup \{v\}$,   $G := G \setminus \{v, w\}$ (double-vertex removal)
     else
       choose $v \in V(G)$ uniformly,
       $C := C \cup \{v\}$,   $G := G \setminus \{v\}$ (single-vertex removal).
3. Output $C$ as Vertex Cover.
```

We describe the underlying ideas of the algorithm. If the graph has at least one leaf, the vertex adjacent to the leaf gets a cover mark, and these two vertices, along with their incident edges, are removed. The idea not to choose leaves for the cover is sometimes called *domination* and is present in many different approximation algorithms and search heuristics for VC (or, from a different viewpoint, independent set); see, e. g., [21,6]. Otherwise, a vertex is picked uniformly for the cover and only this single vertex and its incident edges are removed. In the first case, the graph is reduced in same manner as with the original Karp–Sipser algorithm for maximum matchings. At the first instance where there are no leaves left, our approach starts to behave differently. By definition, KS-VC outputs a valid vertex cover.

Let Phase 1 end at the first point of time when G has no more leaves (note that new leaves can still be created afterwards), in other words G corresponds to the "core" as defined by Bauer and Golinelli [2]. We summarize a main result by Aronson et al. [1]:

Theorem 1 (*Aronson et al. [1]*)**.** *Let $c < e$. Then at the end of Phase 1 of KS-VC, G is w. h. p. a collection of vertex-disjoint cycles.*

We are ready to present the rigorous supplement to the study by Bauer and Golinelli [2]. Despite being a simple consequence from the previous theorem, the following result does not seem to have been stated explicitly so far (at least in the TCS community). A connection between Karp–Sipser and minimum VC is drawn by Gamarnik et al. [10], however, only the possibility of a $(1 + o(1))$-approximation is noticed in the interesting domain $c < e$.

Theorem 2. *Let $c < e$. Then KS-VC finds an optimal vertex cover w. h. p.*

Proof. We analyze the first phase and the rest of the run separately. Let G^* be the graph remaining after the end of the first phase. Each optimal VC for the graph $G \setminus G^*$ can be converted into the result produced by KS-VC on this subgraph by iteratively moving the cover marks from leaves to their neighbors. All edges incident on vertices from $G \setminus G^*$ are covered in the first phase. Hence, it remains to prove that KS-VC produces an optimal cover on G^*.

By Theorem 1, G^* is a collection of vertex-disjoint cycles w. h. p. Let us assume this to happen. KS-VC covers a cycle of length k by a single execution of a single-vertex removal, followed by $\lceil (k-2)/2 \rceil$ executions of a double-vertex removal, altogether using $\lceil k/2 \rceil$ cover marks. Since the cycles are disjoint, an optimal cover for G^* is produced, in total yielding an optimal cover for G. □

3. Iterated local search

Many analyses of RSHs, most notably EAs, on problems from combinatorial optimization reveal that the heuristics are able to mimic components of problem-specific algorithms with a certain probability [11,23,17]. Often this results in expected polynomial runtimes to find optimal or at least good approximate solutions to the problem. Compared to problem-specific algorithms, a loss of polynomial factors in the runtime seems to be a fair price to pay for the wide applicability of the heuristic. Of course, if the problem at hand is well understood and tailored algorithms are available, one would probably prefer to apply the tailored algorithm. Still, it is interesting how RSHs compete with problem-specific algorithms on well-studied problems from combinatorial optimization since such analyses improve our understanding of the working principles of heuristics on realistic problems.

Most of the above-mentioned runtime analyses up to now consider the worst case from a class of problems rather than average-case models. As seen before, the KS-VC algorithm is itself a heuristic, which performs extraordinarily well in the average-case model of sparse random graphs if $c < e$. We now turn our view to more classical search heuristics such as EAs and local search, which have already been analyzed on certain VC instances [8,9]. The well-known (1+1) EA maintains search points from $\{0, 1\}^{|V|}$, i. e., each bit decides whether the corresponding vertex is picked for the cover or not. The "fitness" of a search point is just the size of the current cover, or a penalty value greater than $|V|$ if no valid VC is encoded. The (1+1) EA creates a new tentative solution by flipping each bit of the current cover with probability $1/|V|$. If the new solution has at most the same fitness value as the current one, the new solution is accepted as the current solution, otherwise it is rejected. This procedure is repeated until some stopping criterion is satisfied.

We see that the (1+1) EA has the ability to change many bits in a step but is more likely to perform local steps changing only few bits. If at most two bits, chosen uniformly at random, are allowed to flip, we arrive at a randomized local search (RLS) algorithm. This RLS variant has been investigated by Giel and Wegener [11] and Neumann and Wegener [17] for the maximum matching problem and the minimum spanning tree problem, respectively. In both cases, it turns out that a runtime behavior competitive with the (1+1) EA can be proved with a significantly simpler analysis. Also for other combinatorial optimization problems, it is believed that the search behind globally searching algorithms like the (1+1) EA is driven by quite local steps that flip only few bits and that the more global steps do not constitute a real advantage but only complicate the analysis. Of course, there are artificial examples where a considerable number of bits has to be changed to improve the current search points, e. g., the functions with large Hamming cliffs studied by Droste et al. [5]. However, to the best of the author's knowledge, all runtime analyses of simple EAs in combinatorial optimization focus on neighborhoods of small size.

Let us consider the VC problem more closely and assume that either one or two bits, i. e., vertices, are chosen uniformly at random to be flipped. Elitist search heuristics like the (1+1) EA and RLS only go from valid to valid covers and never increase cover size. Hence, the steps of size 1 are only accepted if they flip a bit from 1 to 0, i. e., conceptually remove a cover mark from a vertex. Steps of size 2 may remove two cover marks or swap a cover mark on a vertex with a previously unmarked vertex. If the two vertices involved in such a swap are not connected by an edge, we already obtain a cover of smaller size by only removing the cover mark from the first vertex.

Motivated by the latter observation, we decide to go a step further towards hybridizations of different randomized search heuristics. One well-known approach is called *iterated local search* [13], which combines "blind" and relatively general stochastic search heuristics with greedy local search. Typically, such algorithms refine the solutions obtained by the more general search heuristic by greedily following local search steps as long as there is a better solution in the neighborhood of the local search. (If worsenings are accepted, the approach might be extended to variable-depth local search; see [20] for a recent runtime analysis.) For the greedy local search procedure, a small neighborhood is preferred, most often the Hamming neighborhood. In terms of vertex cover problem, this means that we follow improving steps that only remove cover marks from the vertices until there is no such step left. Only then the search heuristic tries steps flipping two bits. Such steps can solely be successful if they swap the cover marks of two adjacent vertices since, as mentioned above, otherwise an improvement would already have been realized by the greedy procedure removing a single cover mark.

Inspired by the preceding considerations, we define a search heuristic which can be seen as an iterated local search algorithm combining RLS in a neighborhood of size 2 with a greedy local search algorithm. The search heuristic studied here is simply called *Iterated Local Search (ILS)* and defined as Algorithm 2. It starts from the full cover. As long as there is a vertex with all neighbors covered, this vertex is immediately removed from the cover, which is the greedy aspect. Otherwise, an edge is chosen uniformly. If swapping its endpoints in and out the cover still leads to a valid cover, the swap is accepted. The swaps can be considered as the "blind" steps changing the cover without leading to an immediate improvement.

Algorithm 2 (*ILS*).
```
1. $C := V(G)$.
2. Repeat forever
     If there is a vertex $v \in C$ with all neighbors in $C$ then
        choose a such a vertex, say $v$, uniformly, and set $C := C \setminus \{v\}$
        (vertex-removal operation)
     else
        choose $\{v, w\} \in E(G)$ uniformly; assume w. l. o. g. that $v \in C$,
        if $w \notin C$ and $(C \setminus \{v\}) \cup \{w\}$ is a cover then set $C := (C \setminus \{v\}) \cup \{w\}$
        (edge-swap operation).
```

Like many local search algorithms, ILS does not necessarily find an optimum. Consider a complete bipartite graph with unequally sized subsets in the bipartition and at least two vertices in the smaller subset. If ILS happens to remove all vertices of the smaller subset from the cover, there will be no possible swap operations. Still, VC on bipartite graphs is polynomial-time solvable [4].

For the following analyses, it is crucial to note that ILS never includes a vertex and all its neighbors in the cover when an edge swap is executed. The reason is that a vertex-removal operation would be possible before this edge swap. If the conditions for the cover marks to be swapped are satisfied, we call an edge *selectable*.

3.1. Sparse random graphs and the case $c < 1$

In this subsection, we investigate the performance of the ILS heuristic in the $G(n, c/n)$ model. As mentioned above, the idea is to relate the random choices of the heuristic to an optimal algorithm, in this case KS-VC. It will turn out that ILS is able to reproduce an important subset of the decisions made by KS-VC with polynomially small probability, implying that ILS finds minimum VCs in the domain $c < 1$ with good probability in polynomial time. This result is not obvious since our heuristic only allows local steps and is unable to explore connected components by complete enumeration.

We need a technical lemma.

Lemma 1. *Let G be a random graph according to $G(n, c/n)$ with $c < 1$. Then w. h. p., all induced subgraphs of G on $O(\log n)$ vertices contains $O(\log n)$ edges.*

Proof. Let γ be a constant which can be chosen arbitrarily large, in particular larger than the constant in the first O-term. We consider all subsets of vertices of size exactly $\gamma \log n$ and the event that the induced subgraph on these vertices contains at least $\gamma^2 \log n$ edges. In the end, a union bound will be applied.

The expected number of edges in an arbitrary subgraph of size $\gamma \log n$ equals $\mu := \binom{\gamma \log n}{2} \cdot (c/n) \le (c\gamma^2 \log^2 n)/n$. Using Chernoff bounds with $(1+\delta)\mu = c\gamma^2 \log n$, the probability of having at least $c\gamma^2 \log n$ edges in the subset is at most

$$\left(\frac{e}{1+\delta}\right)^{(1+\delta)\mu} \le \left(\frac{e \log n}{n}\right)^{c\gamma^2 \log n}$$

The number of subsets of size $\gamma \log n$ equals $\binom{n}{\gamma \log n} \le (en/(\gamma \log n))^{\gamma \log n}$. Hence, the probability that there is such a subset containing at least $\gamma^2 \log n$ edges is at most

$$\left(\frac{en}{\gamma \log n}\right)^{\gamma \log n} \left(\frac{e \log n}{n}\right)^{c\gamma^2 \log n}$$

Choosing $\gamma > e^2$ and $\gamma > 1/c$, the last expression is $o(1)$. □

We are ready to state the announced positive result regarding ILS.

Theorem 3. *Let $c < 1$. For every constant $\epsilon > 0$, ILS finds an optimal vertex cover in polynomial time with probability $1-\epsilon-o(1)$.*

Proof. We assume G to have connected components (CCs) of maximal size $O(\log n)$ and $O(\log n)$ edges in each CC. Moreover, we assume that each CC is unicyclic, i. e., there is at most one cycle in each CC. Using standard results on random

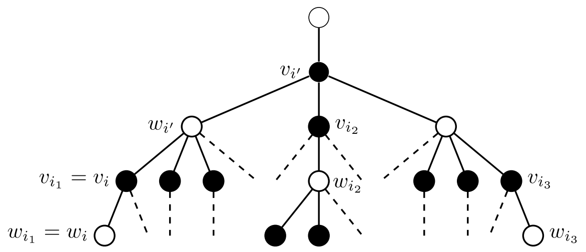

Fig. 1. Example: edge $e_{i'} = \{v_{i'}, w_{i'}\}$, the subtree of $v_{i'}$ and the edges $e_{i_j} = \{v_{i_j}, w_{i_j}\}$ with all these edges in correct state. The v_{i_j} have distance 1 or 2 from $v_{i'}$. Covered vertices are filled.

graphs [3, p. 105] and Lemma 1, the probability that the three properties are satisfied together is $1 - o(1)$. Moreover, with probability at least $1 - \epsilon/2$, the longest cycle has length $O(1)$ [3, p. 117]. All four assumptions hold with probability at least $1 - \epsilon/2 - o(1)$.

In the following, we consider the CCs of G separately and study the probability that ILS to a sufficient extent "simulates" the behavior of KS-VC on this CC. Let a component C^* be fixed. All our following ideas go back to the case of cycle-free CCs, so let us temporarily assume that C^* is a tree. Then KS-VC only executes double-vertex removals, i. e., it chooses edges incident on leaves. We have the freedom to determine the random order according to which KS-VC selects these edges. Therefore, let a root vertex $r^* \in V(C^*)$ for the connected component be fixed arbitrarily and let us assume that KS-VC always chooses a leaf of maximal distance to r^*. Denote by $e_1 = \{v_1, w_1\}, \dots, e_k = \{v_k, w_k\}$ the edges from C^* chosen in this order by KS-VC, i. e., $v_1, \dots, v_k$ are covered. Since a tree structure is assumed, $v_1, \dots, v_k$ is in fact a valid and optimal vertex cover w. r. t. C^*. Let $N(v_i)$ denote the set of neighbors of v_i and note that $w_i \in N(v_i)$. Since $v_1, \dots, v_k$ is a cover, we have $V(C^*) = \bigcup_{i=1}^{k}(\{v_i\} \cup N(v_i))$.

While analyzing ILS, we concentrate on the edges $e_1, \dots, e_k$ and their neighbors. Let e_i, $1 \le i \le k$, be called *correct* w. r. t. a current solution of ILS if only its v-vertex is chosen, *wrong* if only its w-vertex is chosen and *complete* otherwise. The case of both vertices unchosen cannot happen as ILS always maintains valid covers. Moreover, it is crucial that ILS never increases the number of cover marks in a CC. We will show that with probability $n^{-O(1)}$, ILS is able to correct the k edges and, in a manner explained below, also their neighbors, or to arrive at an equally good cover iteratively by a sequence of swap and removal operations. Since the waiting time for such a sequence can be estimated by means of a geometric distribution and since at most n CCs must be corrected, an expected time of $n^{O(1)}$ to find an optimal cover will follow. By Markov's inequality, the time is $n^{O(1)}$ with probability $1 - \epsilon/2$, altogether an optimal cover is found with probability at least $1 - \epsilon - o(1)$.

We distinguish between two cases. Each edge $e_i = \{v_i, w_i\}$, $1 \le i \le k$, is either adjacent to a leaf in the original graph G, or adjacent to a leaf only after at least one double-vertex removal of KS-VC. In the first case, the edge can always be corrected by either removing the w-vertex from the cover (if the edge was complete) or a swap operation (if the edge was wrong). Let p_i be the parent of v_i and note that all neighbors $y \in N(v_i)$ except p_i must be leaves since w_i is a leaf of maximal depth. After the correction of e_i, all covered vertices in $N(v_i) \setminus \{p_i\}$ (and possibly further ones) will undergo removal operations by ILS. Restricted to the induced subgraph on $N^*(i) := \{v_i\} \cup N(v_i) \setminus \{p_i\}$, we have already obtained the same cover as KS-VC, which is, in fact, an optimal cover for this subgraph; we say that $N^*(i)$ has been corrected. Moreover, unless v_i is involved in a swap operation, the number of cover marks in the subtree rooted at v_i never increases. For this to happen, a swap operation would be required choosing an edge not involving v_i and having one vertex in and one vertex outside the subtree, implying a cycle on the whole graph, which has been ruled out by assumption.

Now let us study the case that e_i becomes incident on a leaf only after some steps of KS-VC. Consider the deepest ancestor of v_i lying on one of the edges $e_1, \dots, e_k$. Let i' denote the index of this edge and observe that $i' > i$. If $v_{i'}$ has s children, we investigate for each child the subtree rooted at the child and, in this subtree, all edges from $e_1, \dots, e_k$ having a v-vertex of smallest depth. In particular, e_i is such an edge. Let $e_{i_1} = e_i, e_{i_2}, \dots, e_{i_s}$ be all these edges (see Fig. 1 for an example). Due to the choice of these edges, each v_{i_j}, $1 \le j \le s$, has distance either 1 or 2 from $v_{i'}$. Distance 1 corresponds to a child of $v_{i'}$ being also in the cover produced by KS-VC, distance 2 corresponds to an uncovered child. KS-VC processes the e_{i_j}-edges before $e_{i'}$. Hence, let us suppose that $N^*(i_1), \dots, N^*(i_s)$ are correct in the current cover of ILS, i. e., these sets contain the same cover marks as would be chosen by KS-VC; different neighborhoods $N^*(\ell)$, $\ell < i'$ and $\ell \notin \{i_1, \dots, i_s\}$, need not be correct at this moment (any more). In order to correct $N^*(i')$, it can be necessary for ILS to (1) apply a swap operation to $e_{i'}$ and (2) apply removal operations to some children of $v_{i'}$. Both operations remove cover marks from some children of $v_{i'}$ and are only guaranteed to lead to a valid cover if all edges incident to these children are covered.

However, this is guaranteed by our choice of the edges e_{i_j} since we use all edges from $e_1, \dots, e_k$ with the v-vertex at distance 1 or 2 from $v_{i'}$. Concluding, if we derive sufficient conditions for the correction of $N^*(i')$ before any of the edges incident on $v_{i_1} \cup \dots \cup v_{i_s}$ are touched by ILS again, we obtain a cover of optimal size in terms of the subgraph rooted at $v_{i'}$. A sufficient condition is that $e_{i'}$ is chosen for a swap operation before this happens to the edges incident on $v_{i_1} \cup \dots \cup v_{i_s}$. Since all v_{i_j} must have distance at most 2 from $v_{i'}$, the probability is at least $1/|N_3(v_i')|$, where $N_3(v_i')$ denotes the 3-neighborhood of v_i', i. e., all vertices of distance at most 3 from v_i'. The probability of creating an optimal cover in C^* by an appropriate sequence of swap/removal operations is therefore at least $\prod_{i=1}^{k} 1/|N_3(v_i)|$. It remains to show that the last product is polynomially

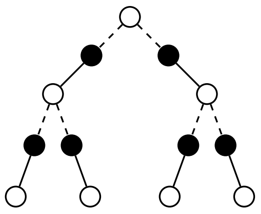
Fig. 2. FoolingTree for $k = 2$ with deep-end edges drawn solid and optimal VC marked by filled vertices.

bounded. Let us instead study $\prod_{v\in C^*} 1/|N_3(v)|$, i. e., we consider even all vertices in C^*. It is easy to see that this product is minimal in regular graphs. Let a denote the number of edges in C^* and c denote the cardinality of C^*. Hence, the size of a regular 3-neighborhood is $(a/c)^3$. We have to bound $(c/a)^{3c}$ from below. As $a = O(\log n)$ and $c = O(\log n)$ is assumed, the term is minimal for $a = ec$, hence $(c/a)^{3c} = 2^{-O(\log n)} = n^{-O(1)}$, which completes the proof for the case that all components are cycle-free.

Our argumentation can be extended to the case of unicyclic components as follows. Let $c_1, \dots, c_\ell$ be the unique cycle in the considered CC C^*. Then C^* is made up of the cycle and $r \le \ell$ disjoint trees $T_1, \dots, T_r$ rooted at the vertices $c_{i_1}, \dots, c_{i_r}$. We again have the freedom to determine the random decisions of KS-VC. Suppose that it chooses as many edges from the trees $T_1, \dots, T_r$ as possible before it considers edges belonging to the cycle. Let E^* be the set of edges incident on any of the c_{i_j}, $1 \le j \le r$, except those edges belonging to the cycle itself. Recalling our argumentation from above, it happens with probability $n^{-O(1)}$ that ILS optimizes all trees $T_1, \dots, T_r$ and sets all vertices on edges from E^* to the same state as KS-VC. If the state of these vertices never changes before the cycle is optimized, we are done. Since the cycle size is $O(1)$, there is a sequence of $O(1)$ swap/removal operations by ILS which leads to the same cover as KS-VC chooses for the cycle (with possibly more than half of the vertices of the cycle chosen). Since there is altogether a number of $O(1)$ edges that we assume not to be touched before the cycle has been optimized, our asymptotic estimations from the last paragraph are not affected. □

3.2. Trees with large connected components

The previous results relied on the fact that connected components are of logarithmic size w. h. p. In this section, we study the behavior of ILS on a sparse graph of maximum degree 3 having a single connected component and show a superpolynomial lower bound on its runtime. Previous results showing that randomized search heuristics fail to efficiently find minimum VCs were available only for dense graphs or graphs with large maximum degrees [8,9].

Our example called FoolingTree is defined on $n = 2^{k+2} - 3$ vertices. The graph is the subdivision obtained from a complete, rooted binary tree by replacing each edge with a path of length two (see Fig. 2 for an example). Hence, there are $2^{\lceil i/2\rceil}$ vertices at depth $0 \le i \le 2k$. Any VC must include vertices from at least every other level.

Fact 1. *The unique optimal VC for* FoolingTree *chooses all vertices of odd depth.*

There are many VCs being by one vertex away from optimality, e. g., the VC choosing all vertices of even depth. In our analyses, we concentrate mainly on the so-called *deep-end* edges between levels $2i - 1$ and $2i$, $1 \le i \le k$, i. e., the deeper edges from the paths of length two; all other edges are called *high end* (see also Fig. 2). Each deep-end edge has a unique upper neighbor (vertex) in a high-end edge. This neighbor is either the root or endpoint of another deep-end edge. Since all vertices except the root are endpoint of exactly one deep-end edge, the configuration of the deep-end edges together with the root is sufficient to specify any vertex cover. We call a configuration of a deep-end edge *correct* if only its odd-level vertex is chosen, *wrong* if only its even-level vertex is chosen and *complete* otherwise (the empty case cannot occur in valid vertex covers). The optimal vertex cover sets all deep-end edges correctly. In the following, we conceptually restrict the tree to the deep-end edges. From this perspective, we denote the set of deep-end edges below a deep-end edge e along with e itself as the *subtree rooted at e* and denote the upmost deep-end edges in the subtree as the two *children of e*.

We justify why FoolingTree is fooling ILS. Let us consider a configuration of ILS where at least one deep-end edge is wrong. To correct the edge by an edge-swap operation of ILS, the configuration of the two children of the deep-end edge is crucial. Only if the two children are not wrong, the edge is selectable and the swap results in a valid VC; otherwise an edge in between the deep-end edge and its children would be uncovered. Let us assume for a moment that the two children are correct. Then there are two choices among the deep-end edge and its children that decrease the number of correct edges but only a single choice increasing this number. Hence, given that a wrong deep-end edge can be corrected, there seems to be a tendency towards more wrong edges.

We make these ideas precise. We give each deep-end edge a depth, defined by the number of deep-end edges from the root to the edge itself. Fix a valid VC and a deep-end edge e. In the subtree $T(e)$ of deep-end edges rooted at e, we define the following potential function denoted by $P(e)$: consider the set of edges in $T(e)$ that are (1) wrong, (2) selectable, and (3) have only wrong ancestors in $T(e)$. If this set is empty then $P(e) := 0$. Otherwise, $P(e)$ is the maximum depth (w. r. t. e) of these edges, increased by 1. The first aim is to show that $P(e)$ has a strong tendency to increase in the run of ILS on the FoolingTree instance. Later, this result will be “amplified” in order to show that ILS needs expected superpolynomial time to optimize FoolingTree. In the following, we call a step of ILS *relevant* for a subtree if it chooses a selectable edge from this tree for an edge-swap operation. We denote by $d(e)$ the maximum value $P(e)$ can take, i. e., $d(e)$ is by one larger than the depth of $T(e)$ (where only deep-end edges are counted).

Lemma 2. *Let e be a deep-end edge and consider its current $P(e)$-value. If $P(e) > 0$ then, with probability at least $1/2$, the $P(e)$-value increases to its maximum $d(e)$ before it reaches 0; the conditional expected number of relevant steps for this is at most $12d(e)$. From a current value $P(e) = d(e)$, the probability to reach $P(e) = 0$ before falling back to $P(e) = d(e)$ is at most $2^{-\Omega(d(e))}$.*

Proof. We apply the Gambler's Ruin Model [7] to the P-value. At $P(e)$-value 0, the gambler is ruined; when $P(e) = d(e)$, where $d(e)$ denotes the depth (measured in deep-end edges) of $T(e)$, the gambler has won the game.

We will show that p, the probability of increasing the $P(e)$-value, can be bounded from below by at least $2/3$. By the definition of $P(e)$, we consider the maximum depth of a wrong, selectable edge having only wrong ancestors. Moreover, ILS can perform only a single swap operation in a step. Together this means that $P(e)$ can decrease by at most 1 in a step. Since increases by more than 1 are possible, the Gambler's Ruin Model provides a pessimistic estimation. With an initial capital of d, the probability of the gambler's ruin is bounded from above by $((1-p)/p)^d \le 2^{-d}$ regardless of the adversary's capital. This proves the last claim of the lemma, and estimating $d \ge 1$, also the first part of the first claim of the lemma. The expected duration of the game (the number of relevant steps of ILS is only smaller) is bounded from above by $d(e)/(2p-1) \cdot (1-(1-p)/p)^{-1} \le 6d(e)$, implying that the conditional expected duration under the condition of the gambler's gain is at most $12d(e)$. This proves the second part of the first claim.

We are left with the claim $p \ge 2/3$. Consider the deep-end edge e^* that determines the current $P(e)$-value, and let e_1 and e_2 denote its (deep-end) children. We already argued why there is only a single way (namely choosing e^*) of decreasing the potential and why the decrease is at most 1. The observation is that both e_1 and e_2 must be correct and selectable. If any of these, w. l. o. g. e_1, was wrong, the high-end edge in between e_1 and e^* would be left uncovered after applying a swap operation to e^*, contradicting its selectability. If, w. l. o. g. e_1, was complete, all three vertices on the path formed by e_1 and its high-end predecessor edge would be covered, contradicting the definition of ILS. Hence, both e_1 and e_2 are correct and, since their parent is wrong, applying a swap to, w. l. o. g., e_1 still results in a vertex cover with a wrong selectable edge e_1 having only wrong ancestors. Since the $P(e)$-value uses the maximum depth of such edges, the swap increases it by at least 1. Hence, $p \ge 2/3$ follows. □

In the forthcoming theorem, we let the potential increase simultaneously for the complete tree and several of its subtrees. Since Lemma 2 assumes a positive initial potential, we set up some sufficient conditions to increase a zero potential.

Lemma 3. *Let e be a deep-end edge with $P(e) = 0$. If its upper neighbor vertex is covered, then there is a single edge-swap operation leading to $P(e) > 0$.*

Proof. To establish $P(e) > 0$, we have to create, by a single edge-swap operation, a deep-end edge in $T(e)$ that is wrong, selectable and has only wrong ancestors in $T(e)$. We start by investigating the status of e. Trivially, e has no ancestors in $T(e)$. Since $P(e) = 0$, we conclude that e is not selectable or not wrong. Since its upper neighbor is covered, e cannot be complete. Otherwise, all three vertices on the path from this upper neighbor to the lower vertex of e would be covered, and ILS would have performed a vertex-removal operation with respect to the upper vertex of e. Thus e is correct if it is not wrong. In the case that e is correct, applying a swap to e results in a wrong edge which is still selectable. As this edge has only wrong ancestors in $T(e)$, we have $P(e) > 0$ then.

We are left with the case that e is wrong but not selectable. We consider a longest path of deep-end edges, starting with e and containing only wrong edges. We have already argued why this path contains at least e itself. Let e^* denote the last edge on the path, where $e^* = e$ is possible. If e^* did not have any children then e^* would be selectable, contradicting the assumption $P(e) = 0$.

Consequently, e^* has two deep-end children (edges) in $T(e)$. Since e^* is not selectable, at least one of these edges is not wrong. If this non-wrong edge was complete, we would again obtain a fully covered path of length three in contradiction to the definition of ILS. Hence, there is at least one child e' of e^* that is correct. Since the upper neighbor of e' (namely, the lower vertex of e^*) is covered, e^* is selectable. Applying a swap to e^* results in a wrong edge which is still selectable. As this edge has only wrong ancestors in $T(e)$, we have $P(e) > 0$ after the swap operation. □

We need the following consequence of the preceding two lemmas. As long as a deep-end edge can be turned wrong, the probability of observing it in correct state decreases exponentially with the depth of the subtree below the edge. Later, this will produce a long waiting time for the correction of the parent of the edge.

Lemma 4. *Let e be a deep-end edge whose upper neighbor is covered. Once $P(e)$ has reached its maximum value $d(e)$, the probability of e being correct is at most $2^{-\Omega(d(e))}$ in every following time step before the upper neighbor of e loses its cover mark for the first time.*

Proof. We assume that $P(e) = d(e)$. Moreover, in all following considerations, we assume the upper neighbor of e to be covered since nothing is left to show when this cover mark has been lost for the first time. By assumption, Lemma 3 is in force. Together with Lemma 2, this means

1. if the $P(e)$-value has reached 0, there is a probability of $\Omega(1)$ to increase its maximum $d(e)$ before reaching 0 again and
2. the probability of reaching $P(e)$-value 0 from value $d(e)$ before falling back to $d(e)$ is $2^{-\Omega(d(e))}$.

We consider a modified process that, after an occurrence of $P(e) = 0$, surely returns to $P(e) = d(e)$ before reaching $P(e) = 0$ again. For any $t \geq 0$, let us look into the tth time step of the modified process after an occurrence of $P(e) = d(e)$. If the probability of observing $P(e) = 0$ at this time step is at least some value p, then the probability of reaching $P(e) = 0$ from $P(e) = d(e)$ before falling back to $d(e)$ is also at least p. Hence, by the above second property, the probability of observing $P(e) = 0$ at time t in the modified process is $2^{-\Omega(d(e))}$. By the first property, we know that $P(e) = 0$ holds in the real process for an expected number of $O(1)$ time steps before the $P(e) = d(e)$ is reached again. Hence, the probability of observing $P(e) = 0$ is at most $O(1) \cdot 2^{-\Omega(d(e))} = 2^{-\Omega(d(e))}$ in any time step after the first occurrence of $P(e) = d(e)$.

Since $P(e) = 0$ holds if e is correct, the probability of e being correct is also at most $2^{-\Omega(d(e))}$ after the first occurrence of $P(e) = d(e)$. □

We are ready to prove our theorem.

Theorem 4. *The expected optimization time of ILS on* FoolingTree *is superpolynomial.*

Proof. Initially, all three vertices on the upmost two edges in the FoolingTree instance are chosen. With probability at least 2/3, the first run of the greedy vertex-removal procedure of ILS removes a vertex at depth 1 rather than the root from the cover. Then the root is covered, the cover is non-optimal and Lemma 3 is in force with e being the upmost deep-end edge. Applying Lemma 2 under this condition, we obtain that with probability $\Omega(1)$, we have at least once a path consisting of only wrong deep-end edges from the top to the bottom of the tree. The remainder of this proof shows that it takes expected superpolynomial time to correct all edges on this path.

Let us take a closer look at the path of wrong deep-end edges, in the following denoted by $e_1 (= e), e_2, \dots, e_k$, where e_{i+1} is a deep-end child of e_i, $1 \leq i \leq k-1$. By e'_{i+1}, we denote the other deep-end child of e_i. The idea is to study the subtrees $T(e'_j)$, $2 \leq j \leq k/2$, and to utilize that many of these reach a large potential before ILS has corrected the last $k/2$ edges on the path. Note that it is impossible to correct e_i unless e'_{i+1} and e_{i+1} are both correct (the complete case will be impossible again). Therefore, the upper neighbors of the e'_{j+1}, $1 \leq j \leq i$, (which are in fact the lower vertices of the e_j, $1 \leq j \leq i$) are covered before e_i is corrected. This will allow us to apply Lemma 4. Starting from a wrong path $e_1, \dots, e_k$, let C_i denote the random time until e_i is corrected for the first time. In the following, we show for $1 \leq i \leq k/2$ that $E(C_i) = 2^{\Omega(k)} \cdot E(C_{i+1})$. Since $k = \Omega(\log n)$, we obtain $E(C_1) = 2^{\Omega(\log^2 n)}$, i. e., a superpolynomial lower bound on the expected optimization time.

To show the claim $E(C_i) = 2^{\Omega(k)} \cdot E(C_{i+1})$, we assume that all subtrees $T(e'_j)$, $2 \leq j \leq k/2+1$, have reached their maximal potential at least once before $e_{k/2}$ is corrected. Using Lemma 4 and the fact that different subtrees are treated independently by ILS, we obtain a probability of $2^{-\Omega(k)}$ that e'_{i+1} is correct at the first instance where $e_{i+1}, \dots, e_k$ are all correct. If e'_{i+1} is not correct, Lemma 2 yields a probability $\Omega(1)$ of reaching maximal potential at least once in the subtree $T(e_{i+1})$ before e_{i+1} is correct again. Hence, e_{i+1} has to be corrected in expectation $\Omega(1) \cdot 2^{\Omega(k)}$ times from a completely wrong path (with possibly different $e_{i+2}, \dots, e_k$) before e_i is corrected for the first time. This proves the claim.

We still have to justify the assumption that all subtrees $T(e'_j)$, $2 \leq j \leq k/2+1$, have reached maximal potential at least once before $e_{k/2}$ is corrected. We start our considerations at the time where the path $e_1, \dots .e_k$ is completely wrong. Using Lemma 2 and Markov's inequality, every subtree $T(e'_i)$ reaches maximal potential in at most $O(k^2)$ relevant steps with probability at least $1-1/k$. Hence, after this number of relevant steps, all subtrees have reached their maximal potential with probability at least $(k/2)/k = 1/2$. Each step is relevant with probability at least $1/|E| = \Omega(1/n)$. Hence, using Chernoff bounds, $O(k^2 n)$ steps suffice with probability $\Omega(1)$ to reach maximal potential in all subtrees $T(e'_j)$, $2 \leq j \leq k/2+1$. On the other hand, by Lemma 2, we also need $2^{\Omega(k)}$ trials with probability $1-2^{-\Omega(k)}$ to correct all edges $e_{k/2+1}, \dots, e_k$. Since there is at most a single choice for a swap operation to decrease the potential of a subtree, the time to correct all those edges is at least $n2^{\Omega(k)}$ with probability $1-o(1)$, according to Chernoff bounds. Altogether, the probability of reaching maximal potential in all those subtrees before $e_{k/2}$ becomes correct is $\Omega(1)$. Since all assumptions together still hold with probability $\Omega(1)$, the unconditional expected optimization time is superpolynomial. □

Conclusions

We have studied the behavior of randomized search heuristics for minimum vertex cover in sparse random graphs according to the $G(n, c/n)$ model and supplemented previous experimental analyses. At first, we have rigorously proven that the problem can the solved to optimality for $c < e$ using a modification of the Karp–Sipser algorithm called KS-VC. Afterwards, a hybrid heuristic called ILS was investigated. For $c < 1$ it reproduces the decisions of KS-VC with a good probability in polynomial time. However, it fails badly already on graphs consisting of a single tree component of maximum degree 3. Our analyses provide insight into the behavior of randomized search and present methods for its analysis. At the same time, they illustrate principles of hybridizations of problem-specific greedy algorithms and randomized search heuristics.

References

[1] J. Aronson, A. Frieze, B. G. Pittel, Maximum matchings in sparse random graphs: Karp–Sipser revisited, Random Structures and Algorithms 12 (2) (1998) 111–177.
[2] M. Bauer, O. Golinelli, Core percolation in random graphs: a critical phenomena analysis, The European Physical Journal B 24 (3) (2001) 339–352.

[3] B. Bollobás, Random Graphs, 2nd edition, Cambridge University Press, 2001.
[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd edition, MIT Press, 2001.
[5] S. Droste, T. Jansen, I. Wegener, On the analysis of the (1+1) evolutionary algorithm, Theoretical Computer Science 276 (2002) 51–81.
[6] I. K. Evans, Evolutionary algorithms for vertex cover, in: Proc. of Evolutionary Programming VII, in: LNCS, vol. 1447, Springer, 1998, pp. 377–386.
[7] W. Feller, An Introduction to Probability Theory and Its Applications, vol. 1, 3rd edition, Wiley, 1968.
[8] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, C. Witt, Approximating covering problems by randomized search heuristics using multi-objective models, in: Proc. of GECCO 2007, AMC Press, 2007, pp. 797–804.
[9] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, C. Witt, Analyses of simple hybrid evolutionary algorithms for the vertex cover problem, Evolutionary Computation 17 (1) (2009) 3–19.
[10] D. Gamarnik, T. Nowicki, G. Swirscsz, Maximum weight independent sets and matchings in sparse random graphs. Exact results using the local weak convergence method, Random Structures and Algorithms 28 (1) (2005) 76–106.
[11] O. Giel, I. Wegener, Evolutionary algorithms and the maximum matching problem, in: Proc. of STACS'03, in: LNCS, vol. 2607, 2003, pp. 415–426.
[12] A. Hartmann, M. Weigt, Statistical mechanics perspective on the phase transition in vertex covering of finite-connectivity random graphs, Theoretical Computer Science (265) (2001) 199–225.
[13] H. H. Hoos, T. Stützle, Stochastic Local Search: Foundations and Applications, Morgan Kaufmann, 2004.
[14] C. Horoba, D. Sudholt, Running time analysis of aco systems for shortest path problems, in: Proceedings of the 2nd International Workshop on Engineering Stochastic Local Search Algorithms, SLS 2009, in: Lecture Notes in Computer Science, vol. 5752, Springer, 2009, pp. 76–91.
[15] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations, Plenum, 1972, pp. 85–103.
[16] R.M. Karp, M. Sipser, 1981. Maximum matchings in sparse random graphs, in: Proc. of FOCS'81, pp. 364–375.
[17] F. Neumann, I. Wegener, Randomized local search, evolutionary algorithms, and the minimum spanning tree problem, Theoretical Computer Science 378 (1) (2007) 32–40.
[18] F. Neumann, C. Witt, Runtime analysis of a simple ant colony optimization algorithm, Algorithmica 54 (2) (2009) 243–255.
[19] M. Pelikan, R. Kalapala, A.K. Hartmann, Hybrid evolutionary algorithms on minimum vertex cover for random graphs, in: Proc. of GECCO'07, ACM Press, 2007, pp. 547–554.
[20] D. Sudholt, Hybridizing evolutionary algorithms with variable-depth search to overcome local optima, Algorithmica, 2011, in press (doi:10.1007/s00453-009-9384-2).
[21] R.E. Tarjan, A.E. Trojanowski, Finding a maximum independent set, SIAM Journal on Computing 6 (3) (1977) 537–546.
[22] I. Wegener, Simulated annealing beats metropolis in combinatorial optimization, in: Proc. of ICALP'05, in: LNCS, vol. 3580, 2005, pp. 589–601.
[23] C. Witt, Worst-case and average-case approximations by simple randomized search heuristics, in: Proc. of STACS'05, in: LNCS, vol. 3404, 2005, pp. 44–56.
[24] C. Witt, Greedy local search and vertex cover in sparse random graphs (extended abstract), in: Proc. of TAMC'09, in: LNCS, vol. 5532, Springer, 2009, pp. 410–419.