Analysis of the (1 + 1)-EA for Finding Approximate Solutions to Vertex Cover Problems

Pietro S. Oliveto, *Student Member, IEEE,* Jun He, *Member, IEEE,* and Xin Yao, *Fellow, IEEE*

***Abstract*—Vertex cover is one of the best known NP-Hard combinatorial optimization problems. Experimental work has claimed that evolutionary algorithms (EAs) perform fairly well for the problem and can compete with problem-specific ones. A theoretical analysis that explains these empirical results is presented concerning the random local search algorithm and the $(1+1)$-EA. Since it is not expected that an algorithm can solve the vertex cover problem in polynomial time, a worst case approximation analysis is carried out for the two considered algorithms and comparisons with the best known problem-specific ones are presented. By studying instance classes of the problem, general results are derived. Although arbitrarily bad approximation ratios of the $(1+1)$-EA can be proved for a bipartite instance class, the same algorithm can quickly find the minimum cover of the graph when a restart strategy is used. Instance classes where multiple runs cannot considerably improve the performance of the $(1+1)$-EA are considered and the characteristics of the graphs that make the optimization task hard for the algorithm are investigated and highlighted. An instance class is designed to prove that the $(1+1)$-EA cannot guarantee better solutions than the state-of-the-art algorithm for vertex cover if worst cases are considered. In particular, a lower bound for the worst case approximation ratio, slightly less than two, is proved. Nevertheless, there are subclasses of the vertex cover problem for which the $(1+1)$-EA is efficient. It is proved that if the vertex degree is at most two, then the algorithm can solve the problem in polynomial time.**

***Index Terms*—Combinatorial optimization, computational complexity, evolutionary algorithms, vertex cover, worst-case approximation.**

I. Introduction

EVOLUTIONARY algorithms (EAs) are randomized search heuristics that have been widely used for solving combinatorial optimization problems since the 1970s [2]. However, the field of the analysis of the time complexity of EAs is fairly new, spawning in the 1990s with the analysis of the basic $(1+1)$-EA on simple pseudo-boolean functions [3]. These functions were artificially created to understand what characteristics of a problem may make its optimization easy or hard for an EA. The behavior of the (1 + 1)-EA on toy problems such as Onemax [3], on Trap Functions [4], or on plateaus of constant fitness [5] was examined. Besides proving whether the algorithm is efficient or not in optimizing the problems, these efforts have led to the introduction of a range of techniques [6] and to the design of a general Markov chain framework [7] for the analysis of EAs. Building upon this first basis, the (1 + 1)-EA has been analyzed on linear functions [8] and on quadratic polynomials [9]. Nowadays its analysis is possible for combinatorial optimization problems with "practical" applications, such as maximum matching [10], [11], the minimum spanning tree problem [12], the partition problem [13], and the subset sum problem [14]. In these papers a new technique, known as drift analysis [14], [15], has proved to be a useful tool, often (but not only) for finding exponential lower bounds. A recent survey describing the available results is in [16].

In this paper, the (1 + 1)-EA is analyzed for the vertex cover problem, a well known NP-Hard combinatorial optimization problem having practical applications in fields such as networking and scheduling. In general, randomized search heuristics, such as EAs, are problem-independent algorithms designed to perform fairly well on large classes of problems. Usually they are used when there are not enough available resources for the construction of a specialized algorithm for solving a specific problem. Hence, it is not generally expected that an EA would outperform a problem-specific algorithm, far less being likely when optimizing NP-Hard problems. However, experimental studies have claimed that EAs are particularly successful on vertex cover instances [17]. In particular, the paper shows that a genetic algorithm (GA) finds better covers than a widely known approximation algorithm (i.e., Vercov) on random graphs with various edge densities [17]. This result appears surprising, and gave the initial motivation for the theoretical work presented in this paper.

It is important to point out that this paper is not an attempt to design competitive algorithms for the vertex cover problem. The main goal is that of understanding the search capabilities of EAs on a difficult combinatorial optimization problem, since it is not possible to analyze EAs on a practical problem with an "unknown" structure. Experimental results have shown that EAs can produce qualitatively good results on a wide range of problems (see for example [2]). In this sense, NP-Hard problems are good test beds for EAs and a theoretical analysis emphasizing the approximation quality that can be guaranteed by an EA and how approximation affects its computation time for vertex cover is highly desired. Even for a nontrivial P-problem, such analyzes can shed light on the relationship between the algorithm and problem characteristics, as demonstrated by Sasaki and Hajek's seminal work on simulated annealing for maximum matching [18].

Manuscript received October 28, 2008; accepted December 21, 2008. Current version published September 30, 2009. This work was supported by an EPSRC grant (EP/C520696/1). Part of the work presented in this paper has previously appeared in the Proceedings of the 2007 Congress on Evolutionary Computation (CEC2007).

The authors are with the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham, B15 2TT U.K. (e-mail: P.S.Oliveto@cs.bham.ac.uk; J.He@cs.bham.ac.uk; X.Yao@cs.bham.ac.uk).

Digital Object Identifier 10.1109/TEVC.2009.2014362

For the maximum matching problem, Giel and Wegener have shown that the (1 + 1)-EA requires exponential expected runtime in the worst case [10]. They have also proved that the (1 + 1)-EA is a polynomial randomized approximation scheme (PRAS) [19] for the problem. This means that it guarantees near-optimal solutions in polynomial time. For the partition problem which is NP-Hard, not only has the (1 + 1)-EA proved to be a PRAS, but it has also been proved to be competitive with problem-specific algorithms in the average case [13]. Understanding when and why improving the current approximate solution requires a considerable increase of the runtime on an NP-Hard combinatorial optimization problems will give greater insights to the application of EAs in practice.

As a first significant step in the analysis of vertex cover, it is necessary to analyze simple versions of EAs. By understanding on which kind of fitness landscape algorithms using only one individual may get trapped and for which reasons this happens, it may be possible to understand which conditions a vertex cover instance needs to satisfy such that populations or operators such as crossover may be useful or necessary for obtaining good solutions. Similar extensions to population-based EAs have been achieved for other combinatorial optimization problems. For example (1 + 1)-EA results have been extended to the $(1 + \lambda)$-EA for the spanning tree problem [12] and to the $(\mu + 1)$-EA for cliques on planar graphs [20].

In this paper, two simple and well-known algorithms are analyzed, the random local search (RLS) algorithm and the (1 + 1)-EA. Since it is not expected that any algorithm solves an NP-Hard problem in polynomial time, their evaluation will be performed in terms of approximate solutions and will be compared with the best and widely used approximation algorithms for vertex cover (i.e., the best ratio known is roughly $2 - (\ln \ln n)/\ln n$ where n is the number of nodes in the graph, which converges to but is always less than two [21]).

Four significant instance classes of the problem are investigated in order to draw some general conclusions about the performance of the algorithms for the problem. Each instance class is designed with characteristics that help to build on the knowledge gained from the analysis of the previous class.

The theoretical analysis of the "Papadimitriou–Steiglitz (PS)" instance class shows how the (1 + 1)-EA has a very similar behavior to that conjectured for the GA using the results of the empirical analysis obtained in [17]. Furthermore, the analysis also shows that there is a constant probability that the algorithm gets trapped on a local optimum. This means that multiple runs are necessary to guarantee the minimum cover is found in polynomial time. However, for the considered graph class, the cover size of the local optimum is roughly doubled compared to that of the minimum cover. This is still a better solution compared to the one found by Vercov [17], and explains the better performance of the EAs for this instance class. However, it is also shown that that this is not the case for variants of the "PS" instance class.

The analysis of the (1 + 1)-EA and the RLS algorithms on a bipartite instance class shows that there exist graphs with similar characteristics to those of the previous instance class that may lead to arbitrarily bad approximation ratios. However, for both instance classes, if multiple runs are considered, then the algorithms find the global optimum quickly.

The analysis of an instance class called $G_{h,l}$ shows a different kind of problem the algorithm may encounter when tackling a vertex cover problem so that, even if a restart strategy is used, the performance may not be improved considerably. As a byproduct of the analysis it is also possible to derive a general result about the (1+1)-EA on any vertex cover problem with vertex degree at most two (i.e., $V_d \leq 2$). The algorithm will optimize every instance with such characteristics in polynomial time but in higher expected time compared to that of problem-specific algorithms for which the problem is trivial ([22], p. 84).

With the aim of proving bad approximation ratios for the (1 + 1)-EA using restarts, it has been possible to build an instance class containing the characteristics of the bipartite graph and those of the $G_{h,l}$ graph. Through the analysis of this instance class it is finally proved that the (1 + 1)-EA cannot guarantee a worst case approximation ratio that is better than that of the state-of-the-art algorithms for vertex cover (i.e., roughly $2 - (\ln \ln n)/\ln n$ [21]).

The rest of this paper is organized as follows. The vertex cover problem, the algorithms considered in this paper, and the previous related work are introduced in Section II. The "PS" instance class is analyzed in Section III. In Section IV, through an analysis of the bipartite instance class, it is proved that restarts are necessary to avoid bad approximation ratios for the (1 + 1)-EA and RLS. An instance class that cannot be optimized even by using multiple runs is analyzed in Section V. Section VI shows that the (1 + 1)-EA is efficient for any graph with vertex degree lower than 3. Finally, Section VII shows that the (1 + 1)-EA and the RLS algorithm cannot guarantee better approximations than problem-specific algorithms for vertex cover.

II. Preliminaries

A. Vertex Cover Problem and the Algorithms

Given an undirected graph $G = (V, E)$, with V being the set of vertices, or nodes, and E the set of edges, the vertex cover problem is that of finding the smallest subset C of V such that for any edge $e \in E$ at least one of its endpoints is in C. All the subsets C of V having, for each edge $e \in E$, at least one of its endpoints in C are called *covers*. All the other subsets of V are *infeasible* solutions. Since vertex cover has been proved to be NP-Hard [23], it is not expected that an algorithm may optimize any vertex cover instance in polynomial time (unless P = NP). An alternative is to accept near-optimal solutions rather than the optimal one. In fact, there exist various approximation algorithms for vertex cover that in polynomial running time return near-optimal solutions. Vercov is a well-known 2-approximation algorithm for the vertex cover problem [23]. This means that whatever the graph instance, Vercov returns a solution which is at most twice the size of the optimal cover.

Vercov starts with an empty cover set and randomly chooses an edge (u, v). Then it deletes all the edges in the graph incident to either u or v and inserts the two endpoints in the cover. The process is repeated until all the edges are removed from the graph.

The two randomized search heuristics considered in this paper are the RLS algorithm and the (1 + 1)-EA. Both the algorithms use bit strings to represent possible solutions to the problem they are trying to optimize. We will also use the term *cover set* to refer to the set of vertices corresponding to the 1-bits in the bit string representing a solution.

Different initializations are examined, varying from an initial *empty cover set* (i.e., the initial solution has no nodes in the cover set), a *uniform distribution* (i.e., each node is inserted in the cover set with probability $1/2$), to an initial *full cover set* (i.e., the initial solution is a cover containing all the nodes of the graph), with the hope of getting an idea of whether one should be preferred to the others. Unless some available information about the problem suggests differently, in practice the uniform distribution is usually used.

For the (1 + 1)-EA and the RLS to be adapted to optimize a vertex cover instance, each subset $C \in V$ is represented by a bit string $(s_1, \ldots, s_n)$ with n being the number of vertices of the graph $G = (V, E)$. For each node v_i belonging to the subset C, the relative bit s_i is set to 1. Otherwise it is set to 0. Given the above representation, a fitness function for the vertex cover problem can be introduced as in [17] and [24]

$$f(C) = \sum_{i=1}^{n} \left(s_i + n(1 - s_i) \sum_{j=1}^{n} (1 - s_j) e_{i,j} \right).$$

Since vertex cover is a minimization problem, the lower the value of $f(C)$, the better the quality of the solution. The first part of the above formula counts the number of nodes in the cover and the second part gives a penalty of n to each uncovered edge $e_{i,j}$ (i.e., an edge connecting the nodes i and j) with $e_{i,j} \in \{0, 1\}$. Since the minimum cover may be at most n for any graph, any *cover* has a better fitness value than that of an *infeasible* solution. The mutation operator used by the RLS algorithm to create new solutions flips one bit per iteration. The (1 + 1)-EA, instead, flips each bit with probability $1/n$. Both algorithms use elitist selection.

Randomized algorithms make random choices during their execution, so they do not perform the same operations in every run, even if the input is the same. Also, they do not necessarily output the same result on a given input if they are run more than once. Hence, the runtime of the algorithm is a random variable. As a consequence, when analyzing randomized algorithms such as RLS algorithms or EAs, the *expected runtime* of the algorithm is used as a measure of their performance.

In particular, if T_f is a random variable measuring the time required by the algorithm to find the solution for a certain function f, the runtime analysis consists of estimating $E(T_f)$, the expected value of T_f.

Sometimes $E(T_f)$ is not sufficient to give an idea of how likely it is that the algorithm will be efficient (i.e., will return the optimal solution in polynomial time with respect to the size of the problem). For this reason, results about $pr(T_f \le t)$, which is the *success probability* given a certain number of steps t, are also desired. As will be shown throughout the paper, both the (1 + 1)-EA and the RLS algorithm may have exponential expected runtimes, but at the same time they may have high probabilities of finding the optimum in a time that is considerably lower. When this is the case, a restart strategy may change an inefficient algorithm into an efficient one.

Let the probability that the optimum is found in time T be a constant c or greater. If the algorithm is restarted $1/c$ times, then the optimum is found in expected time $(1/c) \cdot T$. In practice, however, it may not be trivial to understand when it is the case to stop the algorithm and run it again. This depends on the available knowledge about the problem. A simple restart strategy is that of performing c parallel runs of the algorithm. Then the expected time to find the optimum will still be $(1/c) \cdot T$. A relationship between the number of restarts (i.e., parallel runs) and the probability that the optimum is found may be derived. The probability that a given run of the algorithm does not find the optimum in time T is at most a constant $c' = 1 - c$, and the probability that all k runs do not find the optimum after time T is at most $(c')^k$. Hence with a probability of at least $1 - (c')^k$ the optimum has been found in k runs and $k \cdot T$ total fitness evaluations. The bigger the k, the higher the probability the optimum is found (for example for $k = \sqrt{n}$ the optimum is found in $\sqrt{n} \cdot T$ generations with probability at least $1 - (c')^{\sqrt{n}}$). In the rest of the paper, with *restart strategy* we refer to the parallel run method described above.

In this paper, we are dealing with an NP-Hard problem, and hence we do not expect the algorithms to be efficient on all instance classes. So the performance of the algorithms will also be evaluated according to their worst case *approximation ratio* for the vertex cover problem and the expected runtime required for obtaining a given approximation quality. The worst case approximation ratio of an algorithm A on a minimization problem R is defined as

$$\max_{I \in R} \frac{A(I)}{OPT(I)}$$

where $A(I)$ is the solution obtained by A on the instance I and *OPT(I)* is the value of the best solution of I. The computational complexity results will be defined as a function of the size of the problem (i.e., the number of nodes in the graph) and the time required to find the solution. It is assumed that the reader is familiar with asymptotic notation (see [25]). Furthermore, the reader should refer to [19] for detailed explanations of classical mathematical tools used in the analysis of randomized algorithms. In particular, throughout the paper *Chernoff inequalities* and the *Coupon Collector Theorem* will be frequently used.

The main difference between the RLS algorithm and the (1 + 1)-EA is that the former only flips one bit in each iteration. This means that the RLS algorithm gets stuck on the same local optima as those of a local search algorithm with a neighborhood defined by a hamming distance of 1. For this reason we will also call all such local optima with the exception of the minimum cover *local search covers*.

Since the (1+1)-EA may flip multiple bits in one generation (i.e., each bit is flipped with probability $1/n$), the (1 + 1)-EA cannot get stuck forever on any local optima, and hence on a local search cover. When analyzing the (1 + 1)-EA it will be useful to consider the probability that a certain number of bits flip. The following definition of an i-bit flip will be used in the rest of the paper.

Definition 1 (i-bit Flip): Let n be the number of nodes in the graph to be optimized by the (1 + 1)-EA. When the mutation operator of the (1 + 1)-EA flips i bits in one single generation, for any i such that $0 \leq i \leq n$, then we will say an i-bit flip has occurred.

B. Related Work

This paper is an attempt to understand the capabilities and the limits of EAs when tackling a difficult combinatorial optimization problem. Since the design of efficient algorithms for the vertex cover problem is beyond the scope of this paper, in this section we will only overview previous work focusing on understanding how well or how badly EAs may perform for vertex cover.

Various experimental work have shown that EAs may be a promising approach for the problem (see [17], [26]). The first theoretical analysis of EAs for vertex cover was presented by [24]. It compares an EA without any problem domain knowledge with other two EAs that use different kinds of problem-specific information. However, in this paper, only the convergence of the algorithms is proved while the runtime comparisons are performed empirically.

The first runtime results concerning EAs for vertex cover, which have recently appeared, are those of Friedrich *et al.* [27] and Oliveto *et al.* [1]. The latter work is presented and extended in this paper. Friedrich *et al.* have proved, through an analysis of a bipartite instance class, that the (1 + 1)-EA has a worst case approximation rate which can be arbitrarily bad given polynomial time. In the same paper, it is also proved that a simple evolutionary multiobjective optimizer (SEMO) can optimize the bipartite graph in time $O(n^2 \log n)$. The bipartite instance class will be further discussed in an analysis of a (1 + 1)-EA using restarts in Section IV.

An analysis of the improvements that can be obtained by combining the (1 + 1)-EA with problem-specific algorithms was presented in [28].

III. Papadimitriou–Steiglitz Instance Class

Experimental studies have claimed that EAs are successful on some vertex cover instances [17]. In particular, the empirical results suggest that they can find better approximate solutions than those found by a very well-known approximation algorithm for vertex cover problems (i.e., Vercov [23]), although with higher expected optimization times. It is not sure whether the above experimental observation is generally true and how good the solutions are, and especially what time the EA takes to find such solutions which are supposedly better.

Bäck and Khuri [17] perform experiments comparing a GA with Vercov on random graphs with different edge densities, and the GA produces better results. Furthermore, they show, again empirically, that the GA performs very well on instances of sizes $n = 100$ and $n = 202$ of the PS graph [23] since it finds the optimal cover on average 6 times out of 10 in a runtime of approximately cn^2 where c is a constant greater than zero. In the remaining runs it only reaches a local optimum which, however, is better than the one Vercov finds on this graph. This problem seems to be simple for the GA, although it is not clear what polynomial time is to be expected as the graph size grows, both in the worst and in the average case. Also, the reasons for the GA not finding the optimum in some runs are not clear. We will give a theoretical explanation of the behavior of the evolutionary algorithm on the PS graph and explain the better performance compared to that of Vercov for this instance class. Although the EAs that are analyzed are simpler than the GA used in [17], the results show that the performance of the RLS and the (1 + 1)-EA is asymptotically as good as those conjectured for the GA through the results of the empirical paper. Obviously, both the RLS algorithm and the (1+1)-EA use the same fitness function and representation as those used by the GA in the experimental work.

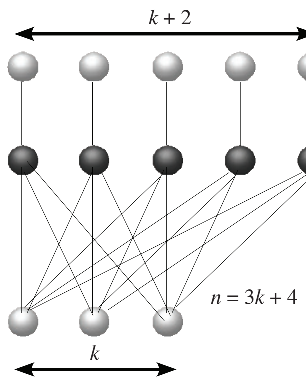

Fig. 1. Optimal cover for the PS graph with $k = 3$. The total number of nodes is $n = 3k + 4$.

The main goal of this section is to understand the behavior of the RLS and the (1+1)-EA on the PS graph and to give an explanation for their better performance compared to Vercov for this instance class. However, it will also be shown that for a slight variation of the instance class (i.e., an instance class called PS-2) the (1+1)-EA does not perform better than Vercov anymore. As a consequence, by the end of the section it will be clear that the best approximation algorithms for vertex cover produce better worst case approximations than the (1 + 1)-EA.

The PS graph and its optimal cover are described in Fig. 1. Each graph of the instance class contains two rows with $k+2$ nodes and a third row with only k nodes. Hence, the total number of rows is $n = 3k + 4$. Each node of the first row is connected to the node of the second row which is in the same column. Each node of the third row, instead, is connected with every row of the second row. As a consequence, the optimal cover is the set containing all the second row nodes. This is true because all the third row nodes have to be in the cover if one or more second row nodes are not in the cover. Furthermore, another $k + 2$ nodes have to be added to the k third row ones to ensure the covering of the edges between the first and the second rows. Hence, a local optimum has at most $2k + 2$ nodes while the minimum cover has $k + 2$ nodes.

This section is divided into three parts. The first part is dedicated to the analysis of the RLS algorithm for the PS graph. The second part analyzes the performance of the (1+1)-EA for a simplified version of the PS instance class called PS-2 and for the original PS instance class. Hence, also the effects that flips of a higher number of bits may produce are considered. The third section analyzes the Vercov algorithm and shows why it produces worse covers for the PS graph than those of the EAs even when the latter algorithms get stuck on a local optimum.

A. Random Local Search

In this section it will be shown that there is a constant probability that the RLS algorithm finds the global optimum. However, the algorithm may get stuck on a local optimum. In that case it will be stuck forever because it only flips one bit at a time. So, even a simple randomized algorithm such as RLS has a behavior which is very similar to the conjectured one of the GA for the PS instance class. The main result of this section is presented in the following theorem.

Theorem 1: The expected time for the RLS algorithm to optimize the PS graph is infinite. With probability at least $k/(2k+2) \geq 1/4$ the algorithm finds the global optimum in time $O(n \log n)$.

The following lemmas consider different initializations of the algorithm. Lemma 1 considers the situation when the algorithm is initialized with all the nodes in the cover set. In Lemma 3, the algorithm is initialized with an empty cover set. Lemma 4, instead, considers the classical uniform distribution initialization.

Lemma 1: If the RLS algorithm is initialized with a full cover set, then the expected time to optimize the PS graph is infinite. With probability at least $k/(2k+2) \geq 1/4$ the algorithm finds the global optimum in time $O(n \log n)$.

Proof: The fitness value of the initial solution is $f(C) = n$ because all the nodes of the graph are in the initial cover. Since the algorithm flips one bit at a time, after the first iteration one node will have been removed from the cover. The obtained solution, being feasible (two nodes need to be removed before an infeasible solution may be obtained), will be accepted by the RLS algorithm because it uses an elitist selection strategy and the new solution will have a fitness value of $f(C) = n-1$. Let E_1 be the event that a node of the third row is chosen before one of the second row. With probability at least $k/(2k+2) \geq 1/4$ event E_1 happens. This is the probability that one of the k third row nodes is chosen in the first step out of the total of all the second and third row nodes (i.e., $k+(k+2) = 2k+2$). If a first row node is chosen instead, then the probability of event E_1 happening becomes higher (i.e., $k/(2k+1)$) because one of the second row nodes, if chosen, will produce an infeasible solution. If event E_1 happens, then no bit flips of second row nodes will be accepted because they would produce an infeasible solution which, having a worse fitness value, will not replace its parent. When all the nodes of the first and the third row have been selected at least once, they will have all been removed and the global optimum will have been found. For the Coupon Collector Theorem [19], the expected time for this to happen is $O(n \log n)$ and the second statement of the lemma is proved. On the other hand, with a probability of $(k+2)/(3k+4)$ a second row node is the first node to be removed from the cover. In such a case a local search cover (i.e., a cover with all the third row nodes and, for each column, one node belonging to either the first or the second row) will be found. At this point the algorithm will be trapped forever since it does not flip more than one bit at a time. Hence the expected optimization time is

$$E(T) \geq \frac{k}{2k+2} O(n \log n) + \frac{k+2}{3k+4} \infty \geq \frac{k+2}{3k+4} \infty = \infty.$$

■

In Lemma 3, the same result will be proved for an empty cover set initialization. The following lemma will be useful for the proof of Lemma 3.

Lemma 2: A bin contains m white balls and l black balls. At each step one randomly chosen ball is removed from the bin without replacement until there is either no white ball or no black ball left in the bin. With probability $m/(m+l)$, all the black balls are removed before all the white ones. With probability $l/(m+l)$, all the white balls are removed before all the black ones.

The proof of Lemma 2 can be found in [27].

Lemma 3: If the RLS algorithm is initialized with an empty cover set, then the expected time to optimize the PS graph is infinite. With probability at least $k/(2k+2) \geq 1/4$, the algorithm finds the global optimum in time $O(n \log n)$.

Proof: First it will be proved that with a probability of at least $k/(2k+2) \geq 1/4$, the algorithm finds the global optimum in time $O(n \log n)$ (i.e., the second statement of the theorem). If the algorithm is initialized with an empty cover set, then any node that is initially chosen to be inserted in the cover set will be accepted. As a consequence, the fitness value improves according to how many edges are covered by the chosen node. A first row node will always be accepted except when the second row node on the same column is already in the cover. Second row nodes, instead, will always be accepted if at least one third row node is not in the cover. The same holds for the third row nodes (i.e., if there is at least a second row node that is not in the cover then any third row node, if chosen, will be accepted). It follows that if *all* the second row nodes are inserted in the cover before *all* the third row nodes, then the algorithm will eventually find the global optimum. The probability of such an event is $k/(2k+2)$. This probability follows from Lemma 2. We consider that there are $2k+2$ balls in the bin, k (i.e., the third row nodes) are white while $k+2$ (i.e., the second row nodes) are black. Our process is equivalent to that of removing one random ball at a time and stopping when we have removed all the balls of one color.

Hence, with probability at least $k/(2k+2)$, *all* the second row nodes are inserted in the cover before *all* the third row nodes. For the Coupon Collector Theorem [19], the expected time is at most $O(n \log n)$. Since, the expected time to remove all the first and the third row nodes left in the cover is also $O(n \log n)$ (from the Coupon Collector Theorem again), the second statement of the lemma is proved. The proof of the first statement follows.

Let event E_1 occur if *all* the third row nodes are inserted in the cover before *all* the second row nodes. If E_1 happens, then there is at least one second row node that is not in the cover. Let v_2 be this node and v_1 be the first row node in the same column as v_2. Let E_2 be the event that v_1 is inserted in the cover before v_2. If events E_1 and E_2 happen, then the RLS algorithm will reach a local optimum and be stuck there forever. This occurs because if v_2 is selected for insertion in the cover, then the fitness value will be incremented by 1, hence the solution will not be accepted by the elitist selection scheme. On the other hand, if v_1 is removed from the cover then an infeasible solution will be obtained (i.e., the edge connecting v_1 and v_2 will not be covered), and hence not accepted.

The probability that event E_1 happens is $p(E_1) = (k+2)/(2k+2)$ (Lemma 2). The probability that event E_2 happens is at least $p(E_2) \geq 1/2$ (i.e., the probability of choosing v_1 out of v_1 and v_2). Hence, with probability at least $1/2\ ((k+2)/(2k+2))$, the algorithm reaches a local search cover in time $O(n \log n)$ due to the Coupon Collector Theorem [19]. Since RLS only flips one bit at a time, it will remain stuck forever on the local optimum. This leads to the infinite expected optimization time. ■

Lemma 4: If the RLS algorithm is initialized with a uniform distribution, then the expected time to optimize the PS graph is infinite. With probability at least $k/(2k+2)$, the algorithm finds the global optimum in time $O(n \log n)$.

Proof: After initialization, half of the nodes of each row are expected to be in the cover set. Let m be the number of nodes of the second and the third row that are inserted in the cover set during the initialization phase. For Chernoff bounds [19] the probability that all the nodes of one of the rows are in the cover after initialization is exponentially small. Hence $m < 2k$ with overwhelming probability. So, just like for Lemma 3, we need to bound the probability that *all* the second row nodes are inserted in the cover before *all* the third row nodes. Since during initialization they have been inserted in the cover set independently and all with the same probability (i.e., 1/2), Lemma 2 can also be applied to this scenario. We consider having $2k+2$ balls in a bin where k are white and $k+2$ are black. First we remove m balls from the bin (i.e., initialization phase where with exponentially high probability $m < 2k$) and then we remove as many balls as required until there are either no black balls or no white balls left. Then, by Lemma 2 the probability that all the second row nodes are in the cover before all the third row nodes is $k/(2k+2)$. At this point for the Coupon Collector Theorem [19], the expected time to reach the optimum is $O(n \log n)$. On the other hand, with probability at least $1/2\ ((k+2)/(2k+2))$ (Lemma 2) the local optimum is found and the algorithm will be stuck forever. This leads to the infinite expected optimization time. ■

The proof of Theorem 1 follows from Lemmas 1, 3, and 4 according to which initialization is considered.

B. (1 + 1)-*EA*

The (1 + 1)-EA has a positive probability of flipping many bits at a time, so it cannot remain stuck on a local optimum forever. However, it may take a long time before it manages to escape.

Since many bits may be flipped in one step, the proofs for the (1 + 1)-EA turn out to be more complicated than those for the RLS algorithm. First, the following simplified instance class, called PS-2, will be considered.

Definition 2 (PS-2 Instance Class): The PS-2 instance class is obtained by removing two nodes from the first row and two nodes from the second row of the PS instance class. All the edges adjacent to the four nodes are also removed from the instance class graphs.

By considering the PS-2 instance class, we obtain considerably simpler proofs which are easier to follow. Calculating the probabilities that all the nodes of the PS-2 graphs are removed from (or inserted in) a given row is easier because the number of nodes in each of the three rows is the same.

The main result of this section is stated in the following theorem.

Theorem 2: The expected time for the (1 + 1)-EA to optimize the PS graph is $2^{\Omega(\sqrt[3]{n})}$. With constant probability the (1 + 1)-EA finds the global optimum in time $O(n \log n)$. Both statements also hold for the PS-2 instance class.

The theorem states that the expected time is exponential. This happens because there is a constant probability that the algorithm finds a local optimum before finding the minimum cover. When this happens, the (1 + 1)-EA requires exponential time to escape from the local optimum and find the global optimum. However, there is a constant probability that the algorithm finds the global optimum in $O(n \log n)$ time (i.e., the second statement of the theorem).

Lemmas 6, 7, and 8 prove, each considering a different initialization, that there is a constant probability that the algorithm finds the minimum cover without getting trapped in any local optimum of the PS-2 graph. After the analysis of the PS-2 instance class, it will be shown how the results can be transferred to the PS instance class for a full cover initialization. The first statement of Theorem 2 will, instead, follow from Lemmas 10 and 11. These two lemmas hold for both instance classes (i.e., PS and PS-2).

For the proof of the following lemmas the concept of an *inversion* will be useful. The definition follows.

Definition 3 (Inversion):

1) Let all the nodes of the second row be in the cover set while $i > 0$ third row nodes are not. Then an *inversion* happens if one mutation inserts all the missing third row nodes in the cover set while removing at the same time at least i second row nodes.
2) Let all the nodes of the third row be in the cover set while $i > 0$ second row nodes are not. Then an *inversion* happens if one mutation inserts all the missing second row nodes in the cover set while removing at the same time at least i third row nodes.

The following lemma about inversions will be useful for the proof of Theorem 2.

Lemma 5: Let all the second-row nodes be in the cover while at least one third row node is not in the cover. Then with probability at least $(1/e) - e^{-\Omega(n)}$ no inversions occur before the global optimum is found in time $O(n \log n)$.

Lemma 5 is general enough to hold for both the PS and the PS-2 instance classes.

Proof: For an inversion to occur, all the third row nodes have to be inserted in the cover while at least the same amount of second row nodes have to be removed. The probability that the third row nodes are all inserted in the cover is highest when there is only one third row node missing from the cover. Let $E^\star$ be the event that the third row node is inserted while at least one second row node is removed. The probability that $E^\star$ happens is bounded above by $1/n$ because one precise third row node has to flip and the probability each node is flipped is $1/n$.

If other j third row nodes are missing from the cover, then the bound has to be multiplied by $1/n^j$, because all the third row nodes missing in the cover have to be flipped at the same time (i.e., each bit flips with probability $1/n$) for an inversion to happen. Hence the above $1/n$ bound holds whatever is the amount of third row nodes missing in the cover.

Let E_1 be the event that another third row node is removed from the cover. The probability that E_1 happens is

$$p(E_1) \geq (k-1)\frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1} \geq \frac{k-1}{en}.$$

Hence, the probability that $E^\star$ occurs before E_1 is less than

$$\frac{p(\star)}{p(\star)+p(E_1)} \leq \frac{1}{n}\bigg/\left(\frac{1}{n}+\frac{k-1}{en}\right) = \frac{e}{e+k-1} \leq \frac{e}{k+1}.$$

It follows that with a probability of at least $1-(e/(k+1))$ event E_1 happens before $E^\star$. Let E_i be the event that i third row nodes are missing and that they are inserted in the cover while at least the same number of second row nodes are removed from the cover. As explained previously, the probability of event $E^\star$ is higher of a factor of $1/n^i$ than that of any other event E_i with $2 < i \leq k-1$. On the other hand, the probability E_1 is only higher of at most a $1/n$ factor than any other 1-bit flips removing a third row node if there are less than $(k-1)$ third row nodes in the cover. Hence, $e/(k+1)$ is an upper bound on the probability of E_1 happening before any event E_i.

It follows that $(1-(e/(k+1)))^{((k+1)/e)-1} \geq 1/e$ is a lower bound for the probability that any event E_i may happen before another $((k+1)/e)-1$ third row nodes have been removed from the cover. At this point the expected time for a third row node to be inserted in the cover is exponential in the number of nodes since at least $(k+1)/e = (n-1)/3e$ third nodes have to flip.

In the mean time the nodes of the first and third row that are still in the cover will be removed. For the Coupon Collector Problem [19], this requires time $O(n \log n)$. ■

Now Lemmas 6, 7, and 8 may be stated and proved.

Lemma 6: If the (1 + 1)-EA is initialized with a full cover set, then with probability at least $(1/6e) - e^{-\Omega(n)}$ it finds the minimum cover of the PS-2 graph in time $O(n \log n)$. With probability at least $(1/6e) - e^{-\Omega(n)}$ a local optimum will be found before the minimum cover.

Proof: If the EA is initialized with all the nodes in the cover, throughout the whole optimization process it will only accept bit flips reducing the cover size or creating a different cover of the same size as the current one. This differs from the optimization process starting with an empty cover set where nodes are added to the cover until a feasible solution has been found.

The proof consists of two parts. The first part proves that with a probability higher than $1/6$ a node of the third row of the graph is removed from the cover before any of the middle row nodes. Then from Lemma 5 it follows that with a probability of at least $(1/e) - e^{-\Omega(n)}$ the algorithm finds the optimal cover. The same idea will be used to prove that a local search cover is found with at least the same probability.

For the first part, the probability that no bits flip can be ignored since it is not influential.

The probability that no bits flip in one generation is $P(0-\text{bit flip}) = (1-(1/n))^n \geq 1/3$. This gives a probability that at least one bit flips of less than $2/3$. The probability that exactly one bit flips in one generation is $P(1-bit\ flip) = n \cdot (1/n)\ (1-(1/n))^{n-1} = (1-(1/n))^{n-1} \geq 1/e$. The conditional probability that *exactly* one bit flips given that *at least* one bit flips is

$$\frac{P(1-bit\ flip)}{1-P(0-bit\ flip)} \geq \frac{1/e}{2/3} \geq \frac{1/3}{2/3} = \frac{1}{2}.$$

Given that a 1-bit flip occurs, the probability that a third row node is chosen rather than a second row node is $1/3$ because each row has got the same number of nodes. Hence, $(1/2)(1/3) = 1/6$ is a lower bound on the probability that at least one node of the third row is selected before a second row node. If a bit belonging to the third row of the graph is chosen in the first step, then any 1-bit flip concerning a node in the second row (i.e., all the nodes of the global optimum) will not be accepted since one of the edges between the second and the third row would become uncovered and the new solution would not be a cover. However, flips of multiple bits may insert all the third row nodes again by removing at the same time some second row nodes (i.e., an inversion occurs). From Lemma 5 with probability $(1/e) - e^{-\Omega(n)}$ no inversions occur before the optimum is found. Hence with a probability of at least $(1/(6e)) - e^{-\Omega(n)}$ the global optimum is found in time $O(n \log n)$.

The second statement of the lemma is proved following the same line of thought. With a probability of at least $1/2$ exactly one bit flips in the first generation. The probability that a second row node is the one chosen is exactly $1/3$. Finally, the probability that no inversions occur before the optimum is found is at least $(1/e) - e^{-\Omega(n)}$. By multiplying the three probabilities, the proof follows. ■

Lemma 7: If the (1 + 1)-EA is initialized with an empty cover set, then with probability at least $(1/2e) - e^{-\Omega(n)}$, it finds the global optimum of the PS-2 instance class in time $O(n \log n)$.

Proof: As in the case of the RLS algorithm, if *all* the nodes of the second row are inserted in the cover before *all* the nodes of the third row, then by just using 1-bit flips the algorithm will reach the global optimum. In the case of the (1 + 1)-EA more than one bit may flip at the same time. From Lemma 5, with a probability of at least $(1/e) - e^{-\Omega(n)}$

the algorithm will find the minimum cover in $O(n \log n)$ time.

Let E_1 be the event that *all* the nodes of the second row are inserted in the cover before *all* the nodes of the third row. We want to find a lower bound on the probability that E_1 happens. Since the $(1 + 1)$-EA may flip more than one bit at a time, some nodes of a row may be exchanged with those of another row.

If the current solution is infeasible, then the bit flip will be accepted if at least the same number of edges are covered by the new solution. If we only consider the second and third rows then, by symmetry of the instance class, the probability that the second row is filled before the third would be exactly $1/2$. This follows from the facts that the number of nodes in each row is the same and each node covers the same number of edges. In the following we will show that the first row influences the process so that the probability the second row is filled before the third row is at least $1/2$. This will follow from the following arguments, which hold for any configuration of the graph if the current solution is infeasible.

1) The probability that a second row node is added to the cover set is higher if the first row is in the instance rather than if it was not there.
2) The probability that a second row node is removed from the cover set is lower if the first row is in the instance than if it was not there.

Point 1) holds because if a first row cover node is exchanged for the second row node on the same column then the step will be accepted since it covers all the third row nodes which are not in the current cover while the first row node only covers one edge. Furthermore, if a third row cover node is exchanged for a second row node, then if the first row node in the same column is not in the cover an extra edge will be covered compared to the case that the first row is not in the graph.

Point 2) holds because a bit flip exchanging a second row cover node for a first row one will not be accepted unless the current solution has all the third row nodes in the cover. If at least a third row node is not in the cover, the second row node covers more edges (i.e., all the ones connecting third row nodes which are not in the current cover) than the first row node which covers only one edge (which was previously covered by the second row node anyway).

Hence, the probability that the second row node is filled before the third row node is higher than $1/2$. Since by Lemma 5 no inversions occur with probability at least $(1/e) - e^{-\Omega(n)}$, the proof follows. ■

Lemma 8: If the $(1 + 1)$-EA is initialized with a uniform distribution, then with probability at least $(1/(4e)) - e^{-\Omega(n)}$ it finds the global optimum of the PS-2 graph in time $O(n \log n)$.

Proof: By Chernoff bounds [19] after initialization neither row has k nodes in the cover with overwhelming probability. Hence, like in Lemma 7 we need to prove that with constant probability *all* the second row nodes are inserted in the cover before *all* the third row nodes. Both rows have the same number of nodes and during initialization each node is inserted in the cover independently with probability $1/2$. Hence, with probability at least $1/2$ after initialization there will be at least the same number of second row nodes in the cover compared to the number of third row nodes. Let x be the number of second row nodes and y the number of third row nodes inserted in the cover during initialization. As discussed above, $x \geq y$ with probability at least $1/2$. Then, the rest of the proof is the same as that of Lemma 7 by considering that $k - x$ second row nodes have to be inserted before $k - y$. From Lemma 7 with probability at least $(1/(2e)) - e^{-\Omega(n)}$ the global optimum is found in time $O(n \log n)$. Multiplying, we get a total probability of finding the minimum cover of at least $(1/(4e)) - e^{-\Omega(n)}$. ■

Since a constant probability to find the minimum cover has been proved, the following corollary of Lemmas 6, 7, and 8 can be stated.

Corollary 1: The $(1+1)$-EA with a restart strategy will find the minimum cover of the PS-2 instance class in expected time $O(n \log n)$.

Now the result of Lemma 6 will be transferred to the PS instance class in the following lemma.

Lemma 9: If the $(1 + 1)$-EA is initialized with a full cover set, then with probability at least $(1/8e) - e^{-\Omega(n)}$, it finds the global optimum of the PS graph in time $O(n \log n)$. With probability at least $(1/6e) - e^{-\Omega(n)}$ a local optimum will be found before the minimum cover.

Proof: The proof follows exactly the same line of thought as that of Lemma 6. The probabilities have to be calculated by considering that the second row has $k + 2$ nodes while the third row only has k nodes.

The conditional probability that *exactly* one bit flips given that *at least* one bit flips is $1/2$.

Given that a 1-bit flip occurs, the probability that a third row node is chosen is at least $(k/n) \geq (n/4)/n = 1/4$, because k nodes may be chosen out of n and $k > n/4$. Hence, $(1/2)(1/4) = 1/8$ is a lower bound on the probability that at least one node of the third row is selected before a second row node. From Lemma 5 with probability $(1/e) - e^{-\Omega(n)}$ no inversions occur before the optimum is found. Hence with a probability of at least $(1/(8e)) - e^{-\Omega(n)}$ the global optimum is found in time $O(n \log n)$.

Since in the first step a second row node is chosen with probability at least $1/2$ $((k + 2)/n) \geq 1/2$ $((n/3)/n) = 1/6$, it follows that with a probability of at least $(1/(6e)) - e^{-\Omega(n)}$ a local optimum is found before the minimum cover. ■

This corollary follows:

Corollary 2: The $(1+1)$-EA with a restart strategy, initialized with a full cover, will find the minimum cover of the PS instance class in expected time $O(n \log n)$.

Lemmas 6, 7, and 8 prove that with a probability of at least $(1/(6e)) - e^{-\Omega(n)}$, the $(1 + 1)$-EA finds the global optimum of the PS-2 instance class in time $O(n \log n)$ (i.e., the second statement of Theorem 2 for the PS-2 graph). Lemma 9, instead, proves a constant probability for the $(1 + 1)$-EA with a full cover initialization of finding the global optimum of the PS graph in time $O(n \log n)$. If this does not happen, then the algorithm will find some local optimum (i.e., a local search cover) before finding the minimum cover. Lemmas 6 and 9 give lower bounds for the constant probability that a local search cover is found for a full cover initialization. In fact,

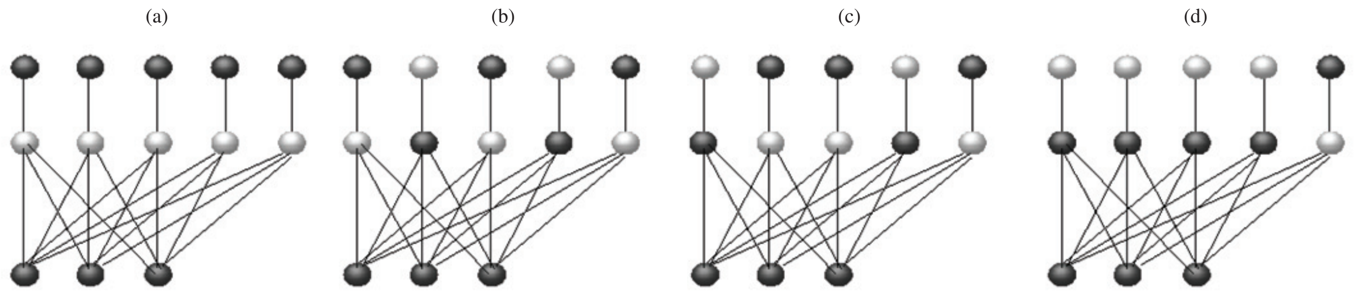

Fig. 2. (a) *Worst* local search cover for the PS graph with $k = 3$. (b) and (c) Different local search covers: all the local search covers, not being the optimal one, have all the third row nodes in the cover, and one node from either of the other two rows for each column. (d) *Best* local search cover with $k = 3$.

for the rest of the proof even an inverse polynomial lower bound on the probability would be sufficient (i.e., to prove the exponential expected runtime to find the minimum cover). For the other two initializations considered in this paper the proof is like that of Theorem 5 in [27]. The proof holds for both the PS and the PS-2 instance classes. Using the multiplicative weight decrease method [12] it is proved that in a phase of at most $12n \log n$ a cover including all the third row nodes is found. Then there is at least an inverse polynomial probability that at least one of the second row nodes has not been chosen in the $12n \log n$ steps, implying that a local search cover has been found. Since the calculations are the same, the complete proof is not reported again here.

The rest of the proof of Theorem 2 is concerned with showing that, if the (1+1)-EA finds a local search cover of the PS graph, then it will require exponential time to escape and find the optimum. In particular, the following will be proved.

1) With probability at least $1/9-o(1)$ the algorithm reaches a local search cover with at least $\sqrt[3]{n}$ first row nodes (Lemma 10).
2) The expected time from such a cover to the minimum one is exponential in n (Lemma 11).

Once the two above results have been proved, the first statement of Theorem 2 will follow. Since the results and the proofs of Lemmas 10 and 11 are essentially the same for the PS and the PS2 instance class, from now on only the original PS graph will be considered.

A local search cover of the PS instance class has all the third row nodes in the cover and one node for each of the $k + 2$ columns belonging to either the first or the second row. Fig. 2 shows four examples of local search covers of the PS instance class with $k = 3$.

Definition 4 (Worst Local Search Cover): The *worst* local search cover is that having all the $k + 2$ first row nodes in the cover, hence none of the second row [Fig. 2(a)].

The *worst* local search cover is the one having maximum hamming distance from the global optimum since, in order to obtain the optimal solution, all the nodes in the cover have to be removed and all the others need to be inserted.

Definition 5 (Best Local Search Cover): A *best* local search cover is that having all the second row nodes in the cover but one [Fig. 2(d)].

A *best* local search cover has the minimum hamming distance from the optimum (i.e., all the third row nodes have to be flipped together with one node each taken from the other two rows; the hamming distance is $k+2$). Fig. 2(b) and (c) are just two other examples showing how one node per column belonging either to the first or the second row has to be in the local search cover together with all the other third row nodes.

Lemma 10: From a *best* local search cover with probability at least $1/9-o(1)$ the $(1+1)$-EA reaches a local search cover with at least $\sqrt[3]{n}$ first row nodes before finding the minimum cover.

Proof: On a *best* local search cover the algorithm has only one node missing in the second row [Fig. 2(d)]. In order to find the global optimum the $(1 + 1)$-EA needs to insert it in the cover (i.e., a necessary condition). If this happens, then the third row nodes may be removed from the cover even by just using 1-bit flips. The node may be inserted in the cover by a 2-bit flip of the only first row node in the cover together with the only second row node not in the cover. Alternatively, the node may be added by two or more bit flips concerning the second row node together with one or more third row nodes. In total there are $k + 1$ possible 2-bit flips and $\binom{k+1}{i}$ possible $i+1$-bit flips with $2 < i \le k$. Let the event that the missing second row node is inserted in the cover be event E_1. If event E_1 happens, then the third row nodes may be "quickly" removed by 1-bit flips and the minimum cover reached in time $O(n \log n)$ (Coupon Collector Theorem [19]). In the following we will prove that with constant probability E_1 does not happen. Afterward it will be shown that if E_1 does not happen, then with probability close to 1 $\sqrt[3]{n}$ first row nodes are inserted in the cover before the minimum cover is found.

The probability of E_1 is

$$p(E_1) \le \binom{k+1}{1}\frac{1}{n^2} + \sum_{i=2}^{k+1}\binom{k+1}{i}\frac{1}{n^{i+1}} = \sum_{i=1}^{k+1}\binom{k+1}{i}\frac{1}{n^{i+1}}$$
$$\le \sum_{i=1}^{k+1}\binom{n/3}{i}\frac{1}{n^{i+1}} \le \sum_{i=1}^{k+1}\frac{(n/3)^i}{i!}\frac{1}{n^{i+1}} = \sum_{i=1}^{k+1}\frac{1}{i!3^i n}$$
$$\le \frac{1}{3n}\sum_{i=1}^{\infty}\frac{1}{i!} = \frac{e-1}{3n} \le \frac{2}{3n}.$$

On the other hand, the probability that at least another node is removed from the second row is greater than

$$(k+1)\frac{1}{n^2}\left(1-\frac{1}{n}\right)^{n-2} \geq \frac{1}{e}\frac{n-1}{3}\frac{1}{n^2} \geq \frac{1}{4en}$$

because a second row node has to flip together with the first row node of the same column.

Let E_2 be the event that at least another second row node is removed before all the second row nodes are inserted. The probability that E_2 happens is greater than

$$\frac{1/(4en)}{1/(4en)+2/(3n)} = \frac{3}{3+8e} \geq \frac{1}{9}.$$

The above calculations prove that if the algorithm was lucky enough to reach the local search cover with smallest distance from the optimum, then there is a constant probability of 1/9 that a local search cover with greater distance from the optimum will be reached (event E_2). In the following it will be proved that if event E_2 happens, then with probability $1-o(1)$ a local search cover with at least $\sqrt[3]{n}$ first row nodes will be reached before finding the optimum. The probability of performing a move adding second row nodes to the cover (p^+) is maximized at distance $d=\sqrt[3]{n}$. This is true because there are $\binom{\sqrt[3]{n}}{j}$ combinations out of $\binom{k+2}{j}$ which add j second row nodes to the cover. The rest of the points in the space with $1<i<\sqrt[3]{n}$ have less combinations leading to an increment of second row nodes (i.e., if there are less than $\sqrt[3]{n}$ first row nodes, then there are also less combinations that if selected will remove them). First we consider only 2-bit flips (moves of length 1 because one first row node is added to the current cover while one second row node is removed, or vice-versa). The probability of adding a second row node (p_1^+), given that only 2-bit flips occur, is at most $\sqrt[3]{n}/n^2 = n^{-5/3}$. The probability of performing a move heading away (p_1^-) from the local search cover with all the second row nodes is minimized at the same position (i.e., $d=\sqrt[3]{n}$). The probability p_1^- is at least

$$\begin{aligned}\frac{(k+2)-\sqrt[3]{n}}{en^2} &\geq \frac{n/3-\sqrt[3]{n}}{en^2} = \frac{n-3\sqrt[3]{n}}{3en^2}\\ &= \frac{n^{2/3}-3}{3en^{5/3}} \geq \frac{n^{2/3}}{4en^{5/3}} = \frac{1}{4en}.\end{aligned}$$

And the probability that p_1^+ happens before p_1^- is at most

$$\frac{1/(n^{5/3})}{1/(n^{5/3})+1/(4en)} = \frac{4e}{4e+n^{2/3}} \leq \frac{4e}{n^{2/3}}.$$

This means that with a probability of at least $(1-(4e/n^{2/3}))$ the opposite event happens. Hence, with a probability of at least

$$\left(1-\frac{4e}{n^{2/3}}\right)^{n^{1/3}} \geq 1-\frac{4e}{n^{2/3}}n^{1/3} = 1-\frac{4e}{n^{1/3}} = 1-o(1)$$

$\sqrt[3]{n}$ negative moves occur before any positive moves.

The above calculations only consider 2-bit flips. Since the probability for a negative move is at least $1/4en$, its expected time is at most $4en$. Then the expected time for $\sqrt[3]{n}$ negative moves to happen is bounded below by $4e\sqrt[3]{n}n = 4en^{4/3}$. By Markov inequality the probability that these $\sqrt[3]{n}$ negative moves happen in a time phase greater than $t=4en^{5/3}$ is

$$P(t \geq 4en^{5/3}) \leq \frac{4en^{4/3}}{4en^{5/3}} = \frac{1}{\sqrt[3]{n}}.$$

Hence, the probability that the $\sqrt[3]{n}$ moves happen before the end of phase $t=4en^{5/3}$ is at least

$$1-\frac{1}{\sqrt[3]{n}} = 1-o(1).$$

Now we will prove that there is at least a constant probability that in the time phase t, no moves of length 2 or greater, heading toward the optimum, happen. At a given position j with $1<j\leq\sqrt[3]{n}$ the probability of a move of length 2 heading toward the optimum is

$$\begin{aligned}P_2 &\leq \sum_{i=3}^{4}\binom{j}{i}\binom{k+2-j}{4-i}\frac{1}{n^4}\\ &\leq \binom{\sqrt[3]{n}}{3}(n-\sqrt[3]{n})\frac{1}{n^4}+2\binom{\sqrt[3]{n}}{4}\frac{1}{n^4}\\ &\leq 2\frac{n^2}{6}\frac{1}{n^4} = \frac{1}{3n^2}.\end{aligned}$$

The above bound also takes into account the probability that the third row cover nodes are exchanged with the missing second row nodes to fill in the second row.

Moves of length greater than 2 have lower probabilities because at least four of the bits considered above have to flip anyway. Hence, the probability that a move of length 2 or greater does not happen in one step is higher than $1-1/3n^2$ and the probability it does not happen in a time phase of length $t=4en^{5/3}$ is at least

$$\left(1-\frac{1}{3n^2}\right)^{4en^{5/3}} \geq \left(\frac{1}{e}\right)^{\Omega\left(\frac{1}{\sqrt[3]{n}}\right)} = 1-o(1).$$

Summing up we get a probability of $1/9-o(1)$ that at least $\sqrt[3]{n}$ first row nodes are in the cover before the global optimum is found. ■

Lemma 11: With probability $1-2^{-\Omega(\sqrt[3]{n})}$ the (1 + 1)-EA does not reach the global optimum from a local search cover with at least $\sqrt[3]{n}$ first row nodes in $2^{c\sqrt[3]{n}}$ steps.

If the current solution is a local search cover with at most $(k+2)-\sqrt[3]{n}$ second row nodes, then for at least one node belonging to the third row to be removed from the cover it is necessary that at least all the second row nodes that are not in the cover get flipped together with the same number of third row nodes. The expected time for such an event to happen is at least $\Omega(n^{\sqrt[3]{n}})$ which is exponential in n. Hence the case of the first row nodes that are exchanged gradually with those of the middle row should be examined to see if the algorithm can reach the global optimum in polynomial time. When all the nodes of the second row have been exchanged with those of the first one, then the third row nodes will quickly be removed from the cover even by just flipping one bit at a time.

In the following, the drift theorem will be introduced as a tool to prove Lemma 11. This proof method was initially introduced by Hajek [29] and afterward was extended to the run-time analysis of EAs by He and Yao [14]. A general description of the technique can be found in [16]. Giel and Wegener introduced a useful extension of the general proof method for proving run-time lower bounds that hold with exponentially high probabilities [10]. Recently, Oliveto and Witt [30] have presented the following theorem which allows considerably simpler calculations. This simplified version will be used to prove Lemma 11.

Theorem 3 (Simplified Drift Theorem): Let X_t, $t \geq 0$, be the random variables describing a Markov process over the state space $S := \{0, 1, \ldots, N\}$ and denote $\Delta_t(i) := (X_{t+1} - X_t \mid X_t = i)$ for $i \in S$ and $t \geq 0$. Suppose there exist an interval $[a, b]$ of the state space and three constants $\delta, \epsilon, r > 0$ such that for all $t \geq 0$:

1) $E(\Delta_t(i)) \geq \epsilon$ for $a < i < b$;
2) $\Pr(\Delta_t(i) = -j) \leq 1/(1+\delta)^{j-r}$ for $i > a$ and $j \geq 1$.

Then there is a constant $c^* > 0$ such that for $T^* := \min\{t \geq 0\colon X_t \leq a \mid X_0 \geq b\}$ it holds $\Pr(T^* \leq 2^{c^*(b-a)}) = 2^{-\Omega(b-a)}$.

The proof of Lemma 11 follows.

Proof of Lemma 11: Let X_t be the random variable denoting the number of second row nodes missing from the cover set (hence the number of first row nodes in the cover set). Furthermore let l be the number of nodes of the first row. So $l = k+2 = ((n-4)/3)+2 = (n/3)+(2/3) = \Omega(n)$. a is set to be equal to one (i.e., $a = 1$), hence any cover containing all the second row nodes, but one, and only one of the first row.

Let the algorithm be in position i when the current solution is a cover with i first row nodes (i.e., $X_t = i$). In the following the drift theorem will be used to prove that exponential time is needed with overwhelming probability before position a (or better) is reached by the (1+1)-EA. $b = \sqrt[3]{n}+1$ so any cover with $\sqrt[3]{n}+1$ first row nodes and $l - \sqrt[3]{n} - 1$ second row ones. Hence $b - a = \sqrt[3]{n} = \Omega(\sqrt[3]{n})$.

Now it remains to check that the two conditions of Theorem 3 hold. The first condition is $E(\Delta_t(i)) \geq \epsilon$ for $a < i < b$.

Let a move of length j occur when exactly j second row nodes are selected to be inserted in the cover and at the same time j first row nodes are selected for removal from the cover. A move will be of length $-j$ when exactly j second row nodes are selected to be removed from the cover together with j first row nodes which are selected for insertion in the cover. Let $p_j(i)$ be the probability that an individual in position i performs a move of length j.

The probability of performing a move of length 1 drifting away from the optimum, given that the process is in position i is

$$p_1(i) = (l-i)\frac{1}{n^2}\left(1-\frac{1}{n}\right)^{n-2} \geq (l-\sqrt[3]{n})\frac{1}{n^2}\left(1-\frac{1}{n}\right)^{n-2} \geq (n/3-\sqrt[3]{n})\frac{1}{n^2}\left(1-\frac{1}{n}\right)^{n-2}.$$

The other moves of length j, with j positive, are omitted as they represent a drift moving away from the optimum. If these steps occur, it will be assumed that the process remains in the current state. This assumption means that the actual process will take longer than the one that is being analyzed. Hence, an exponential runtime for the analyzed process, if proved, will also be an upper bound for the runtime of the actual process.

The probability of performing a move of length -1, heading toward the optimum, given that the process is is position i, is

$$p_{-1}(i) = i\frac{1}{n^2}\left(1-\frac{1}{n}\right)^{n-2} \leq \sqrt[3]{n}\frac{1}{n^2}\left(1-\frac{1}{n}\right)^{n-2}.$$

Since most of the other possible mutations will create infeasible solutions and be rejected, we consider the condition R that a step is relevant, meaning that the step is accepted and changes the current state. The probability of a relevant step p_{rel} is bounded by

$$\frac{1}{n^2}\left(1-\frac{1}{n}\right)^{n-2} \leq p_{\text{rel}} \leq \frac{l+1}{n^2} \leq \frac{n/2}{n^2}.$$

The lower bound holds because in the considered search space (i.e., between 1 and $\sqrt[3]{n}+1$ first row nodes) there are always two bits that if flipped lead to an accepted solution hence it is sufficient to flip these two bits for a relevant step to occur. The upper bound holds because at least two nodes have to be flipped for a step to be accepted (i.e., relevant) and there are $l+1$ couples of such nodes. If more than two bits flip in a relevant step then at least one of the considered $l+1$ couples of nodes must flip and since other bits also flip the probability the event happens is lower than that of the given bound. Hence the upper and lower bounds are correct.

Let $R(i) = (\Delta(i)|R)$ be the random increase of the number of first row nodes in relevant steps for a current state i. Then the contribution of relevant steps of length 1 is

$$\begin{aligned}E(R_1(i)) &= \frac{p_1(i)}{p_{\text{rel}}} - \frac{p_{-1}(i)}{p_{\text{rel}}}\\ &\geq \frac{(n/3-\sqrt[3]{n}-\sqrt[3]{n})(1/n^2)(1-1/n)^{n-2}}{(n/2)(1/n^2)}\\ &\geq \left(\frac{2}{3}-\frac{2\sqrt[3]{n}}{n/2}\right)\frac{1}{e} = \frac{2}{3e} - O(n^{-2/3}).\end{aligned}$$

Let $\Delta^-_{>1}(i)$ be the unconditional increase in the number of second row nodes when more than one second row node is added to the cover, hence steps leading toward the optimum. Then

$$\begin{aligned}E(\Delta^-_{>1}(i)) &\leq \sum_{j=2}^{\sqrt[3]{n}} j \cdot p_{-j} \leq \sum_{j=2}^{\sqrt[3]{n}} j \cdot \binom{i}{j}\frac{1}{n^{2j}}\\ &\leq \sum_{j=2}^{\sqrt[3]{n}} j \cdot \binom{\sqrt[3]{n}}{j}\frac{1}{n^{2j}}\\ &\leq \sum_{j=2}^{\infty} j \cdot \frac{(\sqrt[3]{n})^j}{j!}\frac{1}{n^{2j}} \leq \sum_{j=2}^{\infty}\left(\frac{\sqrt[3]{n}}{n^2}\right)^j\\ &= \sum_{j=2}^{\infty}\left(\frac{1}{n^{5/3}}\right)^j.\end{aligned}$$

Let $r = n^{-5/3}$. Then

$$\sum_{j=2}^{\infty} r^j = r^2 \sum_{j=2}^{\infty} r^{j-2} = r^2 \sum_{j=0}^{\infty} r^j$$
$$= r^2\left(\frac{1}{1-r}\right) = n^{-10/3}\left(\frac{1}{1-r}\right) = O(n^{-10/3}).$$

Hence, the total conditional drift is

$$E(R(i)) \geq E(R_1(i) - \frac{E(\Delta^-_{>1}(i))}{p_{\text{rel}}}$$
$$\geq 2/(3e) - O(n^{-2/3}) - O(n^{-10/3}) \cdot en^2$$
$$= (2/3e) - O(n^{-2/3})$$

and condition 1 is proved.

Since there are at most $(k+2) < n/2$ first row nodes, condition 2 with $\delta = 1$ and $r = 1$ follows from

$$\frac{p_{-j}}{p_{\text{rel}}} \leq \binom{n/2}{j}\frac{1}{n^{2j}} \cdot en^2 \leq \binom{n}{j}\frac{1}{n^j} \leq \frac{1}{j!} \leq \left(\frac{1}{2}\right)^{j-1}.$$

From Theorem 3 the proof follows. ■

Lemmas 6, 7, and 8 prove that with constant probability at least $1/(6e) - e^{-\Omega(n)}$, the (1+1)-EA finds the global optimum before a local search cover of the PS-2 graph. This means that by using a restart strategy, the (1+1)-EA finds the global optimum in time $O(n \log n)$ as stated in Corollary 1. The same result for the PS instance class with full cover initialization is stated in Corollary 2. However, if the algorithm does not find the global optimum straight away, it will end up on a local search cover. Lemma 10 proves that if a local search cover is found, then with probability $1/9 - o(1)$ the (1 + 1)-EA will end up on a local search cover with at least $\sqrt[3]{n}$ first row nodes before finding a local optimum. Finally, Lemma 11 proves that from a local search cover of at least $\sqrt[3]{n}$ first row nodes with probability at least $1 - 2^{-\Omega(\sqrt[3]{n})}$ the global optimum is found in more than $2^{\Omega(\sqrt[3]{n})}$ steps. Theorem 2 for the PS-2 graph with the three initializations and for the PS graph with full cover initialization follows.

C. Vercov

In the last section it has been shown that the RLS algorithm and the (1+1)-EA find the minimum cover of the PS instance class in time $O(n \log n)$ with constant probability. Otherwise, in time $O(n \log n)$ they only find a local search cover of size at most $2k + 2$. The following theorem shows that Vercov always produces a cover of size $2k + 4$. This explains the better performance of EAs compared to Vercov on the PS instance class. Given the similarity of the behavior of the (1 + 1)-EA compared to that conjectured for the GA, it can also be assumed that populations and crossover are not useful on this instance class.

Theorem 4: Vercov always finds a cover of size $2k + 4$ on the PS graph.

Proof: The proof is divided into three points.

1) *All the nodes of the central row of the graph have to be in the cover*. This is because each node in row 1 is connected only with a node of row 2 of the same column. The proof of point 1 follows by considering that Vercov cannot insert a node of row 1 without inserting the node of row 2 in the cover as well.
2) *For each central row node in the cover another node is also inserted belonging to a different row*. This follows by considering again that Vercov inserts two adjacent nodes at a time in the cover, and that no nodes belonging to the central row are connected with each other.
3) *There are no nodes "i" and "j," belonging to the central row and connected to each other by an edge*.

It follows directly from points 1) and 2) that the cover generated by Vercov is *at least* of size $2k+4$, since the nodes in row 2 are $k + 2$. From point 3) it follows that the cover is *at most* of size $2k + 4$ since once there are no edges left Vercov returns the result. ■

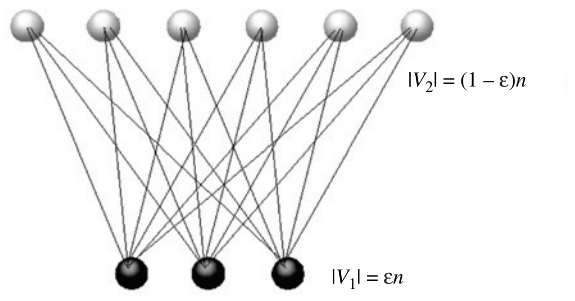

Fig. 3. Bipartite graph with $n = 9$ and $\epsilon = 1/3$. The optimal cover is represented by the dark nodes.

The cover size that the (1+1)-EA can guarantee (i.e., $2k+2$) leads to an approximation ratio of

$$\frac{2k+2}{k+2} = \frac{2k+4}{k+2} - \frac{2}{k+2} = 2 - o(1).$$

Assuming that the similar performance of the GA analyzed empirically in [17] and the (1 + 1)-EA occurs for the reasons described in this section, the above approximation result explains why the GA showed a better performance than Vercov on the PS graph in [17]. However, for the PS-2 instance class, which has four nodes less, the approximation ratio that the (1+1)-EA can guarantee is $2k/k = 2$. Since Vercov guarantees approximations of at most 2, this means that the (1 + 1)-EA cannot produce better approximations in the worst case, far less compared to the best approximation algorithms for vertex cover which guarantee ratios strictly lower than 2. In the next section it will be discussed how the performance of the (1+1)-EA can be much worse.

IV. Bipartite Instance Class

In [27], the (1 + 1)-EA and a global simple evolutionary multiobjective optimizer (SEMO) have been recently analyzed on a bipartite instance class. The instance is depicted in Fig. 3. The instance class contains two sets of nodes V_1 and V_2 of sizes $|V_1| = \epsilon n$ and $|V_2| = (1-\epsilon)n$, respectively. V_1 represents the global optimum while V_2 is the only local optimum. Parameter ϵ determines the difference in size between the two sets V_1 and V_2. In order for $|V_1| < |V_2|$, ϵ has to be $\epsilon < 1/2$.

In [27] it is proved that for $n^{\delta-1} \leq \epsilon < 1/2$ and $\delta > 0$ a constant, the (1 + 1)-EA has an expected optimization time which is exponential to produce an approximation that is better than $(1-\epsilon)/\epsilon$. By changing the value of ϵ, the approximation may be made arbitrarily bad. However, in the following, the (1 + 1)-EA is further analyzed and it is pointed out that, by using multiple runs, the (1 + 1)-EA finds the optimal solution efficiently.

It is worth pointing out that for Koenig's theorem [31], in any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover. Since the maximum matching problem is in P, it follows from Koenig's theorem that also the problem of finding the minimum cover in a bipartite graph is in P. Nevertheless without using a restart strategy, the (1 + 1)-EA requires exponential runtime in the number of nodes of the graph.

Theorem 5: Let $\epsilon = i/n$ with $1 < i < n/2$. The expected optimization time of the (1+1)-EA to find the optimal solution of the bipartite graph is $\Omega(n^i)$. With probability at least $1/(2e) - 2^{-\Omega(n)}$ the (1 + 1)-EA finds the global optimum in time $O(n \log n)$.

Proof: Since $\epsilon n = i$ and $|V_1| = \epsilon n$ it follows that $|V_1| = i$ and $|V_2| = n - i$. It has already been proved in [27] that in time $O(n \log n)$ the (1+1)-EA will find a cover. Furthermore, in a similar manner as for the other instances analyzed in this paper, it will be proved that the (1 + 1)-EA will find a local optimum in expected time $O(n \log n)$. For the bipartite graph there are only two local optima: C_1 containing all the V_1 nodes, and C_2 containing all the V_2 nodes. If the algorithm finds C_1 before finding C_2, then obviously it has found the global optimum. With probability at least 1/2 in $O(n \log n)$ time (Coupon Collector Theorem [19]) and whatever the initialization, *all* the V_1 nodes are in the current cover while *all* the V_2 nodes are not (Event E_1). Event E_1 with a full cover set initialization follows immediately because the probability that in the first step at least a V_2 node is chosen before a V_1 node, and accepted, is greater than 1/2 since $|V_2|/n = 1 - \epsilon > 1/2$ (if more than one node flips and the step is accepted then the probability is greater).

For an empty cover set initialization, we use the same proof idea of Lemma 7 for the PS instance class. We assume the algorithm is given the advantage that the first x nodes of V_2 with $x = |V_2| - |V_1|$ are inserted in the cover before any V_1 node. We call this position $\star$. With this advantage, the probability that E_1 happens is exactly 1/2 if flips of at least two bits removing cover nodes from one set and inserting them in the other could not happen. Without the advantage the probability is greater because, in order to put all the nodes of V_2 in the cover before all the V_1 nodes, it is necessary that at some point of time the number of nodes of the two subsets missing in the cover is at least the same. By considering the bit flips exchanging cover nodes, from position $\star$ on, at each step there are ϵn more cover nodes that can be removed from C_2 and inserted in C_1 than if the sets C_1 and C_2 were equal. If the two sets were equal, the influence of the exchanges of cover nodes from one set to the other would be "symmetrical," leading to a probability of exactly 1/2 that one set was filled before the other. Hence, these extra ϵn C_2 cover nodes lead to a probability greater than 1/2 that C_2 is filled before C_1. The case of uniform distribution initialization is a special case of the empty cover set initialization. So, event E_1 happens with probability at least 1/2.

The conditional probability of finding the global optimum, given that the event E_1 has happened, will be proved to be higher than $1/e$. The probability that the V_2 set is filled by a flip of multiple bits, which removes at least the same number of V_1 nodes, is upper bounded by $1/n$ because at least one precise V_2 node has to be inserted in the cover and the probability this happens is $1/n$. The above probability bound holds in the case with highest probability that the V_2 set has all its nodes but one in the cover. The bound has to be multiplied by $1/n^i$ if other i V_2 nodes are not in the cover. Hence the above probability bound will be considered for the following calculation because it leads to the lowest probabilities of finding the optimum. When there are all the V_2 nodes in the cover but one and all the V_1 nodes are also in the cover, the probability of removing an extra V_2 node is at least

$$P_{V_2} \geq \frac{(1-\epsilon)n - 1}{n}\left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{n/2}{en} = \frac{1}{2e}.$$

Hence the probability an extra V_2 node is removed before one or more nodes are removed from the V_1 set is at least

$$\frac{1/2e}{1/2e + 1/n} \geq \frac{n}{n+2e} = 1 - \frac{2e}{n+2e}$$

and the probability that $n/6$ V_2 nodes are removed before any V_1 node is at least

$$\left(1 - \frac{2e}{n+2e}\right)^{n/6} \geq \left(1 - \frac{2e}{n+2e}\right)^{\frac{n+2e}{2e}-1} \geq \frac{1}{e}$$

which proves the second statement of the theorem because at this point the event that any V_1 node is removed from the cover requires that at least $2n/6 = n/3$ bits flip. The expected time for $n/3$ bits to flip is exponential while the optimal cover will be found in time $O(n \log n)$ (Coupon Collector Theorem [19]).

If, instead, C_2 is found first, then the time to reach C_1 from C_2 has to be analyzed. It has already been proved in [27] that the probability of ending up on the local optimum (i.e., the set C_2 is in the cover and not all C_1) is bounded below by an inverse polynomial. Once C_2 has been obtained, the only accepted moves will be those flipping all the V_1 nodes together with at least the same amount of nodes belonging to V_2. The probability that all the C_1 nodes are flipped together with at least the same amount of V_2 nodes is less than n^{-i}, giving an expected runtime of at least $\Omega(n^i)$. ■

For the RLS algorithms, inversions may not occur because only one bit flips in each iteration. However, the part of the proof of Theorem 5 that the (1 + 1)-EA finds the minimum cover with probability at least $(1-\epsilon) \geq 1/2$ also holds if only 1-bit flips are allowed. However, when the RLS algorithm finds the local optimum it will be stuck forever because it only flips one bit at a time. Now, the following corollary of Theorem 5 can be stated.

Corollary 3: The expected time for the RLS algorithm to optimize the bipartite graph is infinite. With probability $(1-\epsilon)$ it finds the global optimum in time $O(n \log n)$.

The problem the $(1+1)$-EA encounters when optimizing the bipartite graph is that there is a good probability that it finds a local optimum which is not global. If this happens, then exponential time is required for the algorithm to escape. Compared to the PS graph where the size of the local optima is slightly less than twice that of the global optimum, for the bipartite graph the difference in sizes between the local and the global optimum can be made very large by varying parameter ϵ. This is what leads to the very bad worst case approximation ratios of both the RLS algorithm and of the $(1+1)$-EA if no restarts are allowed. The following corollary of Theorem 5 can be stated.

Corollary 4: Both the $(1+1)$-EA and the RLS algorithm using a restart strategy will find the minimum cover of the bipartite instance class in expected $O(n \log n)$ time.

V. Exponential-Time Instance Class

The previous section shows how the worst case approximation ratio of both the RLS algorithm and the $(1+1)$-EA can be arbitrarily bad if only single runs are considered. However, the proofs rely on the fact that the algorithms are not allowed to restart. In fact, by examining the bipartite instance class, the worse the approximation ratio that the algorithm may produce, the higher the probability that the minimum cover is actually found. Hence, if a restart strategy was to be used, then the more "difficult" is the instance because it may lead to a bad approximation, the fewer are the restarts required for the global optimum to be found in time $O(n \log n)$.

A more interesting result in the analysis of the $(1+1)$-EA for vertex cover is that of finding the worst case approximation ratio of the algorithm even when a restart strategy is allowed. Such a result would also be useful for fair future comparisons with population-based EAs. In this section the $G_{h,l}$ instance class is presented to address $(1+1)$-EAs with restart strategies. It will be shown that depending on how the parameters of the instance class are set the problem may be easy or difficult for the $(1+1)$-EA and the results do not change if the algorithm is run many times.

Each graph in the instance class $G_{h,l}$ depends on two parameters l and h, both integers, where $n = l * h$ is the total number of nodes, and l is odd. Each of the $l * h$ nodes is uniquely defined by a couple (i, j) with $1 \leq i \leq h$ and $1 \leq j \leq l$. Each node (i, j) is connected by an edge to the nodes $(k, j-1)$ and to the nodes $(k, j+1)$, with $1 \leq k \leq h$. The unique optimal cover of a graph $G_{h,l}$ has $((l-1)h)/2 = (n-h)/2$ nodes. Fig. 4 depicts a graph $G_{3,5}$ and its optimal cover is represented by the dark nodes.

The idea of analyzing a series of connected bipartite graphs was previously used for proving exponential runtimes for simulated annealing on maximum matching [18] and for the $(1+1)$-EA for the same problem [10]. However, the $G_{h,l}$ instance class is similar but not the same as the instance class used in the previous work.

The following theorem shows what the local search covers look like, and hence helps to get a first idea of what the approximation ratio may be. Afterward, various parameter settings will be considered in separate sections.

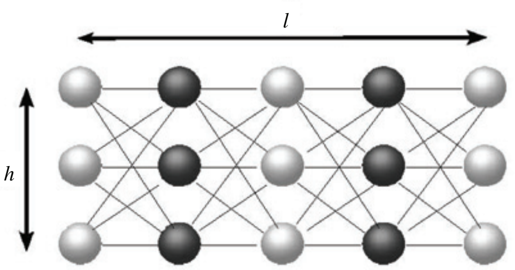

Fig. 4. Optimal cover for the $G_{3,5}$ graph.

Theorem 6: The $(1+1)$-EA finds a cover of at most $\lfloor (2/3)l \rfloor h$ nodes in expected time $O(n \log n)$ for the $G_{h,l}$ instance class.

The following three lemmas prove the theorem for the different initializations considered in this paper.

Lemma 12: If the $(1+1)$-EA is initialized with all the nodes in the cover set, then the expected time to reach a cover of size at most $\lfloor (2/3)l \rfloor h$ is $O(n \log n)$ for the $G_{h,l}$ graph.

For the proof of Lemma 12 the following definitions of *blocked nodes* and of *free nodes* will be useful.

Definition 6 (Blocked Node): A node belonging to a cover is said to be *blocked* if the selection operator does not accept a 1-bit flip of such a node (i.e., by flipping the bit, the new solution would not be a feasible cover anymore).

Definition 7 (Free Node): A node belonging to a cover is said to be *free* if the selection operator would accept the solution obtained by a 1-bit flip of such a node (i.e., by flipping the bit the new solution is a cover of smaller size).

The above definitions can also be extended to columns of nodes. If all the nodes of a column are blocked then the column is said to be *blocked*. Given such definitions, the following properties for covers of the instance class graphs can be described.

1) Each node that is not present in the cover *blocks* $2h$ nodes of the cover: the whole column on the left of the node and the whole column on its right. The far left and the far right columns make an exception by blocking only h nodes each.
2) A node i is *free* only if all the nodes in the columns at its right and at its left belong to the current cover. Since all the nodes in a column are connected with the same nodes of the adjacent columns, if a node is not in the current cover then all the nodes of its column are free.
3) A column is blocked if at least one node belonging to one of its adjacent columns does not belong to the current cover.

Now, by considering the three properties described above, the proof of Lemma 12 is given.

Proof of Lemma 12: Given a cover of the graph, in every group of three adjacent columns at most two can be blocked, hence one must be free (i.e., if the three adjacent columns are all in the cover then the nodes in the middle column are obviously free). It follows that there may be at most $\lfloor (2/3)\, l \rfloor$ blocked columns in total. It is worth pointing out that the first two columns (or the last two columns) of the graph cannot both be blocked at the same time, because if the second column is in the cover then the first is free (and if the column before the last is blocked then the last column is free).

Hence, the selection operator of the $(1+1)$-EA will surely accept 1-bit flips as long as the number of nodes in the cover is larger than $\lfloor(2/3)\,l\rfloor h$. From the Coupon Collector Theorem [19], an upper bound of $O(n \log n)$ can be obtained for the expected time to reach any local search cover. Only 1-bit flips are considered, since flips of higher number of bits may exchange the columns being free and being blocked, or may speed up the described process. In any case they will never augment the cover size. ■

Lemma 13: If the $(1+1)$-EA is initialized with an empty cover set, then the expected time to reach a cover of size at most $\lfloor(2/3)\,l\rfloor h$ is $O(n \log n)$ for the $G_{h,l}$ graph.

When starting with an empty cover set, the characteristics of the nodes that will be accepted by the elitist selection operator are different from those of the accepted nodes when the optimization process starts with a full cover set. This will be true until a cover is obtained. For these reasons the proof of Lemma 13 requires different definitions for free nodes and blocked nodes.

Definition 8 (Blocked Node): A node not belonging to the cover is said to be *blocked* if all its edges connect it with nodes belonging to the cover.

Definition 9 (Free Node): If a node is not blocked, then it is said to be *free*.

The proof of Lemma 13 follows.

Proof of Lemma 13: The algorithm, starting with an empty set, adds any free node it randomly chooses to the cover set until there are no more free nodes. In such a case a cover has been reached because all the remaining nodes that do not belong to the current cover set are blocked.

At the beginning of the process all the nodes are free, since the cover is empty. As the process goes on, when a column has all the column at its right and all the column at its left in the cover, then it is completely blocked. The described process will have a maximum length when all the nodes but one are accepted before a cover is reached (i.e., the columns are filled from right to left or from left to right). In such a worst case event, the nodes of all columns but one will be inserted in the cover, hence they have to be chosen. This case can be compared to the worst case of the Onemax problem, when the process starts with a bit string of all zeros and has to be turned to a bit string of all 1s. The only difference is that the process surely terminates when there is only one column left to be filled at most, hence the expected time is a little lower. So the upper bound of $O(n \log n)$ [32] for the $(1+1)$-EA on Onemax is valid for the expected time of the algorithm to reach a cover in this case (the result follows from the Coupon Collector Theorem [19]). When, instead, some columns or nodes get blocked, the cover will be smaller and the $O(n \log n)$ upper bound still holds.

As stated above, in the worst case the cover that is reached has at most $n-1$ nodes. From such a cover, Lemma 12 guarantees an upper bound of $O(n \log n)$ on the expected time for the algorithm to reach a local search cover. The proof of the lemma follows. Again higher number of bits flipped per generation may only speed up the process since they may exchange the free and blocked columns or add or remove more than one node in a generation. In any case multiple bit flips will never lead to a larger cover. ■

For the proofs of Lemmas 12 and 13 to hold, it is not important that the $(1+1)$-EA is respectively initialized with a full cover set and an empty cover set. For Lemma 12 to hold, it is sufficient that the initial solution is a feasible cover. Then the algorithm will find a solution with at most $\lfloor(2/3)\,l\rfloor h$ nodes in time $O(n \log n)$. On the other hand, the proof of Lemma 13 holds for the $(1+1)$-EA starting with any infeasible solution. These arguments are important to understand how Lemmas 12 and 13 can be extended to the $(1+1)$-EA initialized with a uniform distribution. With such an initialization, before the evolutionary process begins, each node is inserted in the cover set with probability $p = 1/2$. The resulting set of nodes may or may not be a cover. In the former case, Lemma 12 can be applied to prove Corollary 5. In the latter case, Lemma 13 can, be applied. The corollary follows.

Corollary 5: If the $(1+1)$-EA is initialized with a uniform distribution, then the expected time to reach a cover of size at most $\lfloor(2/3)l\rfloor h$ is $O(n \log n)$ for the $G_{h,l}$ graph.

The proof of Theorem 6 follows from Lemmas 12, 13, and, Corollary 5 according to which initialization is considered. Furthermore, since the proof of Theorem 6 also holds if only 1-bit flips are allowed, the following corollary may also be stated.

Corollary 6: The RLS algorithm finds a cover of at most $\lfloor(2/3)\,l\rfloor h$ nodes in expected time $O(n \log n)$.

Theorem 6 ensures that the approximation ratio of the two algorithms may be no more than

$$\frac{\lfloor \frac{2}{3} l \rfloor h}{\frac{l}{2} h} \leq \frac{\frac{2}{3} l}{\frac{l}{2}} = \frac{4}{3}.$$

Although this approximation ratio is far from optimal, it is still better than the one guaranteed by the best known approximation algorithm for vertex cover (i.e., roughly $2 - (\ln \ln n)/\ln n$ [21]). In Section VII, it will be proved that both the $(1+1)$-EA and the RLS are not better worst case approximation algorithms for the problem than the best known problem specific algorithm.

According to the values of the parameters h and l, the optimization of the $G_{h,l}$ may be very difficult or easy for the $(1+1)$-EA. In order to gain a better understanding of what characteristics of a problem may be more or less challenging for the algorithm, the following three sections will consider three different parameter settings with significant importance.

A. When h or l Are Not Constants: $h = l$

In this section, a parameter setting is considered where neither parameter h or l is constant. In particular, h and l will be set such that $h = l$ and $n = h * l$. Hence, $h = l = \sqrt{n}$ and both parameters grow by the same amount as the number of nodes in the graph grows. In the following, it will be proved that with this setting the expected runtime of the $(1+1)$-EA is exponential in the graph size n. The main result of this section is stated in the following theorem.

Theorem 7: The (1 + 1)-EA has an exponential expected runtime $\Omega(n^{\sqrt{n}})$ on $G_{h,l}$ if $h = l$.

Theorem 6 shows that at least a local search cover is found in time $O(n \log n)$. The following three lemmas prove that with exponentially low probability it is the global optimum if parameter l is not constant. The proof of Theorem 7 will follow afterward.

Lemma 14: If the (1+1)-EA is initialized with all the nodes in the cover, then with probability at least $1-(2/3)^{\Omega(l)}$ it finds a local search cover before finding the global optimum.

For the proof of the lemma, the definition of blocked node (i.e., Definition 6) for a full cover set initialization will be useful again. Furthermore, the following concept of *inversion of a blocked column* will be used.

Definition 10 (Inversion of a Blocked Column): Let an *inversion* of a blocked column x be an event where a bit flip of more than one bit inserts in the cover all the missing nodes of the columns adjacent to x and removes the same number of nodes (or more) of the x column from the cover.

The proof of Lemma 14 follows.

Proof of Lemma 14: For the global optimum to be found, all the nodes belonging to columns i, with odd i, have to be removed from the cover. First, the probability that this happens before any even column nodes are removed will be derived. After, it will be proved that if an even column node is removed, then the probability of reaching the global optimum before another local search cover is even lower.

Starting with a full cover set, the probability of choosing an odd column node by flipping one bit is $p_0 = \lceil l/2 \rceil h/lh \leq (l/2+1)/l = 1/2 + 1/l$. Flips of higher number of bits will be treated separately one at a time. In other terms, if, for example, two bits flip and they are accepted by the selection operator, then they will be considered as if one flip happened at a time. Since we are only dealing with the probability that the global optimum is found and not with the required runtime, this abstraction can be made.

At each step, if the most "convenient" odd column is chosen, then four columns are blocked (i.e., for the first step, if a node of the third column is removed, then if nodes of the second and the fourth column were to be chosen, the flips would not be accepted. Hence the first column nodes are destined to be removed also), so at least $\lceil 1/4\, l \rceil$ different column nodes have to be removed before 1-bit flips will guarantee finding the global optimum.

At the next step the probability of choosing another odd column node from a different column out of $X_1 \geq l-4$ columns is $p_1 = \lceil X_1/2 \rceil / X_1 \leq (X_1/2+1)/X_1 = (1/2) + (1/X_1)$. In general the probability of blocking the even columns at step i is $P_i \leq (X_i/2) + 1/X_i = (1/2) + (1/X_i)$, with $X_i = l - 4i$. Hence, the probability in each step is slightly higher than $1/2$ and always lower than $2/3$. The highest probability of choosing the best node at the last step occurs when there are three columns left and two good choices. This leads to a probability of $2/3$, meaning that, for every step, $2/3$ is an upper bound for the probability of removing the first node of each odd column. Hence the probability of finding the global optimum by luckily removing the "right" nodes is less than $(2/3)^{1/4l} = (2/3)^{\Omega(l)}$.

For each of the $l/4$ steps that are necessary for the optimum to be found, we have assumed that once the even columns are blocked then single bit flips will remove the remaining nodes, rather than a bit flip of more than one bit filling the odd columns again while removing some nodes from an even column (i.e., an inversion happens). It is optimistically assumed that if an even column is blocked, an *inversion* of that column will not happen. If it were to happen, then the probability that the algorithm finds the optimum before a local search cover would be lower because another inversion would have to occur. Hence, the $(2/3)^{\Omega(l)}$ bound that the algorithm finds the global optimum before a local search cover also holds if inversions are considered. As a consequence, the probability that a local search cover is found before the global optimum is at least $1-(2/3)^{\Omega(l)}$. ■

Lemma 15: If the algorithm is initialized with an empty cover set, then with probability at least $1-(3/4)^{\Omega(l)}$, the (1 + 1)-EA finds a local search cover before finding the global optimum.

For the proof of Lemma 15, the definition of blocked nodes for empty cover set initialization (i.e., Definition 8) will be useful.

Proof of Lemma 15: All the chosen nodes will be inserted in the cover except when they are blocked. This happens when all the nodes in its two adjacent columns are in the cover. When this happens, the blocked column (i.e., the column in the middle) will be quickly emptied by using 1-bit flips. There is also a probability that before many nodes are removed from the middle column, flips of multiple bits may fill it again by removing contemporaneously some nodes belonging to the two adjacent columns. Since the columns are all of the same size, the probability that this happens is the same for the columns belonging to the minimum cover (i.e., the even columns) and for columns belonging to local search covers. As a consequence, these inversions will not influence the probabilities that will be calculated in the following.

A sufficient condition (not necessary though) for a local search cover to be found is that two adjacent columns are in the cover. In the following we will find the probability that at least two adjacent columns end up in the cover.

We assume that the first column to be filled is an even column (i.e., belongs to the minimum cover). If it were to be an odd column, the probabilities of reaching the minimum cover before a local search one would be lower. Let this column be the ith column. Since all the columns have the same size, the probability that an adjacent column (i.e., column $i+1$ or $i-1$) is filled before the next columns (i.e., $i+2$ or $i-2$) is $1/2$. Let the column that is filled with probability $1/2$ be the column $i+1$. This column does not belong to the optimal cover; hence to reach the local optimum all its nodes have to be removed before the column $i+3$ also gets filled. Column $i+1$ can be removed if the column $i+2$ is filled before column $i+3$. If this happens, then there would be three adjacent columns (i.e., i, $i+1$, and $i+2$) so the middle column nodes would be free to be removed by 1-bit flips. On the other hand, the $i+3$ column is filled before the $i+2$ column with probability $1/2$. It is important to note that this event is independent of the state of the $i+4$ column. The probability is $1/2$ whether the $i+4$ column is full or not. If this happens, then a local search cover will be reached before the global optimum. Summing up, with a probability of at least $1/4$ these four columns are covered in the wrong way. So there is a probability of at most $3/4$ that three columns (i.e., i, $i+1$, and $i+2$) are covered correctly. If this happens, then the same reasoning holds for the remaining $l-3$ columns because of the independence of the $i+4$ column and of the first $i-1$ columns. In total, the probability that the global optimum is found before a local search cover is less than $(3/4)^{\lfloor l/3 \rfloor} = (3/4)^{\Omega(l)}$. ■

By considering that for large enough h with overwhelming probability the algorithm is initialized with at most a certain number of nodes in each column, the corollary of Lemma 15 follows.

Corollary 7: If the algorithm is initialized with a uniform distribution, then with overwhelmingly high probability, the $(1+1)$-EA does not find the global optimum before it finds a local search cover if parameter l is not constant.

Given Lemmas 14 and 15 and Corollary 7, the proof of Theorem 7 follows.

Proof of Theorem 7: Once the EA has found a cover of at most size $\lfloor (2/3)\, l \rfloor h$ which is not the global optimum, the minimum number of bits that need to be flipped in order to leave the current position is $2h$. There are at most $l-1$ possible $2h$-bit flips that would be accepted. Hence, the probability P_{move} of an accepted $2h$-bit flip is at most

$$P_{\text{move}} \le (l-1)\left(\frac{1}{n}\right)^{2h} \le l n^{-2h} \le n^{-h}.$$

Such a probability implies n^h expected generations for an accepted move, which leads to a lower bound for the expected time of $\Omega(n^{\sqrt{n}})$. ■

Theorem 7 shows that the expected time to find the minimum cover of the $G_{h,l}$ instance class is exponential in the number of nodes of the graph. Since the expected time is exponential with high probability, not even a restart strategy will reduce the runtime of the RLS or of the $(1+1)$-EA to reach the global optimum. This result highlights the significance of Theorem 6, from which it follows that an approximation of at most $4/3$ can be achieved in $O(n \log n)$ time for the instance class. Hence, with a relaxation of the exact optimality, the runtime may be considerably reduced.

B. Parameter l Is Constant

If parameter l is constant, then there is a constant probability, although small, that the algorithm reaches the global optimum directly before reaching any local search covers. However, if this does not happen, then the algorithm will need exponential time to escape from any local optima since parameter h is not constant. The main results of this section are stated in Theorem 8 (for RLS) and in Theorem 9 [for the $(1+1)$-EA].

Theorem 8: If parameter l is constant, then the RLS algorithm finds the minimum cover of the $G_{h,l}$ graph with constant probability $2^{-(l/2)}$ in time $O(n \log n)$. The expected optimization time is infinite.

Proof: For simplicity the proof assumes that the algorithm is initialized with all the nodes in the cover. For the algorithm to find the global optimum, it has to remove at least one node of each odd column before it removes one of each even column. The probability that the first node to be removed belongs to an odd column is $p_1 = \lceil (l/2) \rceil / l \ge 1/2$. The probability that the next node belonging to a different column is another odd one is $p_2 \ge \lceil (l/2) - 3 \rceil / l - 3 \ge 1/2$. In general $p_i \ge 1/2$, and the probability that all the odd columns are selected before any even one is at least $2^{-(l/2)}$. The proof that the runtime is $O(n \log n)$ follows from the Coupon Collector Theorem [19]. If one node is chosen from an even column before any of its adjacent columns, then it will be removed from the current cover. If this occurs, since the RLS algorithm only flips one bit per generation, the minimum cover will never be found. Hence, the expected optimization time is infinite. ■

Theorem 9: If parameter l is constant, then the $(1+1)$-EA finds the minimum cover of the $G_{h,l}$ graph with constant probability $(1/e)2^{l/2}$ in time $O(n \log n)$. The expected optimization time is $2^{-\Omega(n)}$.

Proof: The proof is similar to that of Theorem 8, and a full cover set initialization is assumed again. With a probability of at least $2^{-(l/2)}$ at least one node of all the odd column nodes is removed before any even column node. However, for the $(1+1)$-EA the probability that an inversion of a blocked column occurs, leading toward a local optimum, has to be taken into account. For each even column, the probability it is emptied of a number of cover nodes that have been removed from an adjacent column is at most $P_{\text{inv}} \le h/n^3$ because at least two precise adjacent columns have to be inserted in the cover and at least one node of the even column out of h has to be removed. In total at least three bits have to flip. On the other hand, the probability an extra node is removed from one of the two odd columns is at least

$$P_{\text{noinv}} \ge 2(h-1)\frac{1}{en} \ge \frac{2h}{2en} = \frac{h}{en}.$$

So the probability that an inversion occurs before an extra adjacent node is removed is at most

$$\frac{h/n^3}{(h/n^3) + (h/en)} \le \frac{1/n^2}{(1/e) + (1/n^2)} \le \frac{e}{e+n^2} \le \frac{e}{n^2}.$$

So with probability at least $1 - e/n^2$ an inversion does not occur for a given even column before an extra adjacent node is removed. As these adjacent odd column nodes get removed, probabilities for an inversion get lower. Hence, with a probability of at least $1/e$, n^2/e improvements occur before an inversion. Since $(l/2)h < n^2/e$, other improvements are sufficient for the minimum cover to be found, the first statement of the theorem follows.

Since l is constant and h grows with the graph size n, the exponential expected runtime follows from Theorem 7 (i.e., with probability greater than $1 - (2/3)^{l/4}$ a local search cover is found before the minimum cover (Lemma 14) and then the expected time to escape is exponential). ■

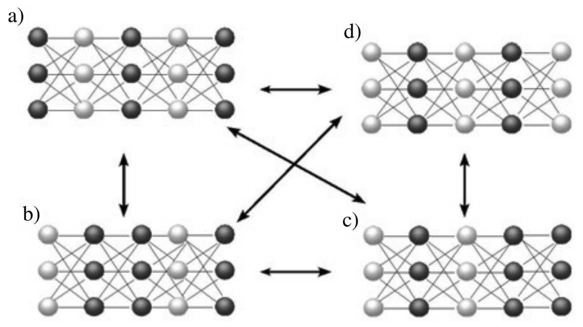

Fig. 5. Description of all the possible accepted steps after the 1+1-EA has reached an almost perfect cover on the $G_{3,5}$ graph. A move from position a) to position b) requires that the leftmost $2h = 6$ bits are flipped [the same applies for an inverse move from b) to a)], while a different combination of the same number of bits allows a move from b) to c). A longer move from a) to c) requires the correct $4h = 12$ bits to flip.

C. Parameter h Is Constant

If h is constant, then once the algorithm has reached a local optimum it will be able to find the global optimum in polynomial time by walking on plateaus of constant fitness. After reaching the end of each plateau, the algorithm will be able to reduce its fitness by h by just using 1-bit flips. The main result of this section is stated in the following theorem.

Theorem 10: The (1 + 1)-EA has a polynomial expected optimization time $O(n^{k+2})$ with $k = 2h$ if h is constant.

The proof of the theorem depends on three points.

1) The (1 + 1)-EA takes expected time $O(n \log n)$ to reach a cover of size $\lfloor (2/3)l \rfloor h$ (Theorem 6).
2) The (1 + 1)-EA takes expected time at most $O((d)^{2h+2})$ to improve each fitness level where $d = l/(2i)$ and i is the number of levels left to be improved before the optimum has been found.
3) The number of fitness levels that need to be improved is at most $\lfloor (2/3)\, l \rfloor - l/2$.

The following lemma is at the heart of the proof. The following two definitions will be useful for its proof.

Definition 11 (Almost Perfect Cover): The smallest local search cover that is not the minimum cover is called an *almost perfect cover*.

Definition 12: Let all the solutions having the same fitness value belong to a set defined by the fitness value. If the fitness value is i, then the set is called *fitness level* of value i.

The most difficult fitness level to improve is that of an almost perfect cover with the highest possible hamming distance from the global optimum. Such a cover for a $G_{3,5}$ graph is depicted in Fig. 5(a).

Lemma 16: The expected time for the (1 + 1)-EA, to reach the optimal cover from an almost perfect cover is $O(n^{k+2})$ with $k = 2h$ where h is constant.

Proof: Once the EA has found an *almost perfect cover*, the only point in the search space having a higher fitness value is that representing the solution to the optimization problem. Thus the selection operator of the algorithm will only accept, as new individuals, the global optimum or another cover having the same size as the current one. All the possible moves having a probability of being accepted for a $G_{3,5}$ graph are analyzed in Fig. 5. The move having a highest probability of occurring on the plateau of almost optimal covers is that flipping the right $2h$ bits while keeping the values of the other $n - 2h$ bits unchanged. This move has a probability of

$$P_{\text{successful}} = \left(\frac{1}{n}\right)^{2h} \left(1 - \frac{1}{n}\right)^{n-2h} \geq \frac{1}{e}\left(\frac{1}{n}\right)^{2h}. \tag{1}$$

Such a probability implies an upper bound for the expected number of generations before such a move occurs of $en^{2h} = O(n^k)$ if h is a constant. Hence, in such a case, the expected time for a move is polynomial in n.

In general, the expected time for a move of length i, drifting toward or away from the global optimum, is bounded above by $O(n^{ik})$. Such a move is called *successful*. A successful move of length i of getting nearer to the optimum has the same probability $p = 1/2$ as one of the same length of moving in the opposite direction. The scenario is very similar to that of a (1 + 1)-EA optimizing a short path problem except that the probability of a successful move in this case is lower. Hence the rest of the proof, presented in the following, is similar to that of Jansen and Wegener for the short path function with constant values (SPC_n) [5].

As shown in (1), the probability of a successful move is at least $(1/e)(1/n^k)$. Then by Chernoff bounds it can be derived that the probability of having less than $c'n^2$ successful steps in cn^{k+2} steps is exponentially small (i.e., $e^{-\Omega(n)}$) with c' being any constant greater than zero.

Now it will be proved that in a typical run these $c'n^2$ successful steps are all of length 1 (i.e., all the moves are obtained by flipping $k = 2h$ bits).

The conditional probability of flipping at least $2k$ bits given that the move is successful is

$$P(2k|succ.) = \frac{P_{2k} \cap P_{\text{succ.}}}{P_{\text{succ.}}} = \frac{\Theta(1/n^{2k})}{\Theta(1/n^k)} = \Theta\left(\frac{1}{n^k}\right).$$

This probability implies typically n^k generations for a move of length 2 on the plateau space. Hence, in a typical run all the $c'n^2$ moves will be of length 1.

Let a successful step moving toward the global optimum be a *positive* move, while one moving in the opposite direction be *negative*. In the worst case, the algorithm will reach the plateau on the farthest away position from the global optimum, which is at a distance $d = l/2$ on the plateau space. Given a successful move, the probability of it being positive is equal to that of it being negative (except for the first step in the considered worst case, i.e., $P_{\text{positive}} \geq 1/2$). Hence $c'l^2$ Bernoulli trials with a success probability of 1/2 may be considered.

Since $|positive| + |negative| = c'l^2$ and $|positive| - |negative| = l/2$, we need at least $(1/2)c'l^2 + (1/4)\, l$ positive steps out of the $c'l^2$ total steps to guarantee a visit to the global optimum.

The probability of having less than $(1/2)\, c'l^2$ positive steps is bounded above by 1/2, and for large enough n and small enough c', the probability of exactly i positive steps is bounded above by $1/(2l)$. So

$$P\left(|pos.| < \frac{1}{2}c'l^2+\frac{1}{4}l\right) = \frac{1}{2}+\sum_{i=\frac{1}{2}c'l^2}^{\frac{1}{2}c'l^2+\frac{1}{4}l-1}\frac{1}{2l} \leq \frac{1}{2}+\frac{l}{4}\frac{1}{2l} = \frac{5}{8}.$$

So with a probability of at least 3/8, cn^{k+2} steps are sufficient to reach the global optimum, giving three expected trials for reaching the end of the plateau (i.e., position c) in Fig. 5. ■

The lemma proves an upper bound of $O(n^{k+2})$ to improve the last fitness level when the length is $d = \lfloor l/2 \rfloor$ (i.e., the distance in number of column exchanges), which is the longest path that can be encountered by the algorithm (i.e., the last fitness level). Another fitness level away from the optimum (i.e., two columns of nodes still have to be removed from the cover before the optimum is reached) the maximum starting distance on the plateau space is $(d/2)$ since an extra block of two consequent adjacent columns has to be present for the fitness to be higher by one. This level requires a time of $O(n/2)^{k+2}$. In general, a path of length d/i requires time $O(n/i)^{k+2}$. Hence the expected time to find the global optimum starting from the worst fitness level is

$$\begin{aligned}\sum_{i=1}^{\lfloor\frac{2}{3}l\rfloor-\frac{l}{2}}\left(\frac{n}{i}\right)^{2h+2} &= n^{2h+2}\sum_{i=1}^{\lfloor\frac{2}{3}l\rfloor-\frac{l}{2}}\left(\frac{1}{i}\right)^{2h+2}\\ &\leq n^{2h+2}\sum_{i=1}^{\lfloor\frac{2}{3}l\rfloor-\frac{l}{2}}\left(\frac{1}{i}\right)^{4}\\ &\leq (n^{2h+2})\frac{\pi^4}{90} = O(n^{2h+2}).\end{aligned}$$

This completes the proof of Theorem 10.

Corollary 8: The expected time for the (1 + 1)-EA to optimize *paths* (i.e., the $G_{h,l}$ graph $h = 1$) is $O(n^4)$.

Since for each step that is accepted $2h$ bits have to flip, the RLS algorithm is not efficient for the $G_{h,l}$ instance class also if parameter h is constant. This highlights the advantage of having a large search neighborhood like that of the (1+1)-EAs mutation operator. Only by using problem-specific information can the performance of the RLS algorithm be improved. The neighborhood size of the RLS could be modified for it to work well for the graph, by letting the algorithm flip h bits with probability 1/2 and $2h$ bits with probability 1/2. Then also, the RLS would be able to perform random walks on the plateau, hence find the minimum cover in polynomial time.

VI. Graphs of Vertex Degree $V_d \leq 2$

Although, the (1 + 1)-EA is not a good approximation algorithm for vertex cover in general, there may be some subclasses of the problem for which the algorithm is efficient. An interesting subclass is that having only vertices with degree $V_d \leq 2$. This subclass of the vertex cover problem is in P and it is easy to design an algorithm that can find the minimum cover in linear time. This section analyzes the performance of the (1 + 1)-EA for this problem. It is shown that the algorithm is efficient on this subclass but not competitive compared to the state-of-the-art problem-specific algorithms [22]. First definitions of *walks, paths,* and *cycles* will be given which are derived from graph theory. These concepts will be useful to state some corollaries of Lemma 16. Then the main result of this section will be stated in Theorem 11.

In graph theory, the following definition holds for *walks* [33]:

Definition 13 (Walk): A *walk* $W_{0,k}$ from v_0 to v_k in a graph $G(V, E)$ is a finite sequence $W = v_0e_{0,1}v_1e_{1,2}v_2 \dots v_{k-1}e_{k-1,k}v_k$. The vertex v_0 is called the *origin* of the walk $W_{0,k}$, the vertex v_k is called the *terminus* of $W_{0,k}$ while the vertices $v_1, \dots, v_{k-1}$ are called the *internal vertices* of $W_{0,k}$. The integer k, which is the number of edges in the walk, is called the *length* of $W_{0,k}$.

Note that in a walk there may be repetitions of vertices and edges. This does not happen in *paths* that are defined as follows [33].

Definition 14 (Path): A walk $W = v_0e_{0,1}v_1e_{1,2}\dots e_{k-1,k}v_k$ of length k on a graph $G(V, E)$ is called a *path* of length k if all the vertices of W are distinct.

The instance considered in Corollary 8 is called a *path* because there is only one set of different edges and vertices of length k.

In the following, an instance class called *Cycle* will be considered.

Definition 15 (Cycle): A *cycle* of length k in a graph $G(V, E)$ is a path of length k with the addition of an edge connecting the vertices v_0 and v_k.
The optimal cover of a *cycle* with n vertices has size $\lceil n/2 \rceil$; hence it has an extra node compared to the optimal cover of a *path* with an odd number of nodes. As a consequence, the only difference for the (1 + 1)-EA in the optimization of cycles compared to paths is that there is one fitness level less to be improved. In other terms, the (1 + 1)-EA requires less time to optimize cycles compared to paths since the problem is a special case of Lemma 16 where $h = 1$ and the *almost perfect cover* has the same fitness value as the minimum cover (i.e., they are both global optima). Hence, the upper bound of $O(n^4)$ also holds for cycles and the following is a corollary of Lemma 16.

Corollary 9: The expected time for the (1 + 1)-EA to find the minimum cover of a cycle with n vertices is $O(n^4)$.

Also, in paths with even number of nodes the last fitness level does not have to be improved. Hence the following corollary of Lemma 16 holds.

Corollary 10: The expected time for the (1 + 1)-EA to optimise *paths* of n vertices, where n is even, is $O(n^4)$.

Now we are ready to state the main result of this section.

Theorem 11: The (1 + 1)-EA finds the minimum cover of any graph with edge degree $V_d \leq 2$ in expected time $O(n^4)$.

Proof: Any graph $G(V, E)$ with degree $V_d \leq 2$ is isomorphic to another graph in which each connected component i with $|V_i|$ vertices is either a *path* or a *cycle* of $|V_i|$ vertices, and $|V| = |V_1| + |V_2| + \cdots + |V_N|$ where N is the number of connected components in G. Hence all the connected components of a graph with $V_d \leq 2$ are paths or cycles.

From Corollaries 8, 9, and 10, the expected time for the (1 + 1)-EA to optimize a path or a cycle with $|V_i|$ vertices is $O(|V_i|^4)$. Hence, the expected time for the algorithm to find

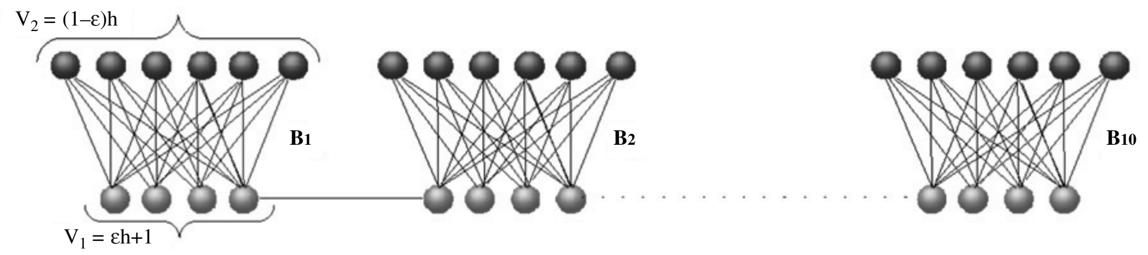

Fig. 6. Description of the $B_{h,l}$ graph of size $n = 100$ with $l = \sqrt{n} = 10$, $b = \sqrt{n} = 10$, $h = b - 1 = 9$ and $\epsilon = 1/3$. The graph consists of $l = 10$ bipartite subgraphs of size $b = 10$ each. Each bipartite subgraph has a set of $V_1 = \epsilon h + 1 = 4$ nodes and a subset of $V_2 = (1 - \epsilon)h = 6$ nodes. The optimal cover is the set of light colored nodes.

the optimal solution is at most $O(|V_1|+|V_2|+\cdots+|V_N|\}]^4) = \sum_{i=1}^{N} O(|V_i|^4) = O(|V^*|^4)$ where V^* is a path of length n. This is true because the sum is maximized when there is only one component with maximal length. ■

VII. Hard-to-Approximate Instance Class

In this section the $B_{h,l}$ instance class is presented and analyzed. $B_{h,l}$ graphs combine both the characteristics encountered in previous subclasses that create optimization difficulties for the RLS algorithm and the (1 + 1)-EA. A bipartite graph, as shown in Section IV, can lead the algorithms to very bad approximation ratios. However, by using a clever restart strategy, the difficulties can be overcome by both algorithms. The $G_{h,l}$ class of graphs analyzed in Section V showed that if l columns of h nodes are connected in a "bipartite" manner, then even multiple runs of the algorithms will not be effective as long as neither l or h are constant in the problem size. The $B_{h,l}$ instance class is constructed by inserting both characteristics in the graphs. Rather than having l columns of length h (as in the $G_{h,l}$ class), l bipartite graphs $(B_1, \ldots, B_l)$ of size $b = h + 1$ each are joined together by an edge connecting the smaller subset. Parameter $b = l = \sqrt{n}$; hence $h = \sqrt{n} - 1$. The total number of nodes is $n = b * l$.

Each bipartite graph has a subset V_1 of size $\epsilon h + 1$ and another V_2 of size $(1 - \epsilon)h$ with $\epsilon < 1/2$. The instance class and its optimal cover, are depicted in Fig. 6.

The following theorem is the main result of this section. It proves that RLS and the (1 + 1)-EA cannot do better in terms of worst case approximation than problem-specific algorithms for vertex cover.

Theorem 12: Let $\epsilon > n^{-1/2+\delta}$, with $0 < \delta < 1/2$ a constant. The expected time for the RLS algorithm to find the minimum cover of the $B_{h,l}$ graph is infinite, while the expected time for the (1 + 1)-EA is exponential in the number of nodes of the graph. With an overwhelming probability, neither the RLS algorithm nor the (1 + 1)-EA find an approximation that is better than $2(1 - \epsilon) - o(1)$ in polynomial time.

The proof of the theorem will be split in two lemmas, one regarding the RLS algorithm and the other regarding the (1 + 1)-EA.

Lemma 17: Let $\epsilon > n^{-1/2+\delta}$, with $0 < \delta < 1/2$ a constant. The expected time for the RLS algorithm to find the minimum cover of the $B_{h,l}$ graph is infinite. With an overwhelming probability the RLS algorithm does not find an approximation that is better than $2(1 - \epsilon) - o(1)$ in polynomial time.

Proof: Let B_i be *optimized* if all its V_1 nodes will be in the cover while all the V_2 nodes will not. If a B_i is *not optimized*, then all its V_2 nodes will be in the cover. In addition, also the V_1 nodes connecting B_i to the bipartite subgraphs B_{i-1} and B_{i+1} may be in the cover. In particular, there will be $l-1$ such V_1 nodes in the worst local search cover (i.e., all the bipartite subgraphs are *not optimized*). These $l - 1$ nodes will be called v^* nodes. Each bipartite subgraph B_i, with $1 \leq i \leq l$, may end up being *optimized* or *not optimized* at the end of the run. The global optimum is found if all the B_i subgraphs are optimized.

Let the RLS algorithm be initialized with all the nodes in the cover.

For a bipartite subgraph B_i to be optimized, it is sufficient that a V_2 node of the subgraph is selected and removed from the cover. If this event happens, since RLS only removes one node per iteration, the removal of no V_1 nodes will be accepted because it would uncover some edge.

The probability that in the first iteration a node from V_2 is selected is

$$P_1 = \frac{(1-\epsilon)hl}{hl+l} = \frac{(1-\epsilon)h}{h+1} < \frac{(1-\epsilon)h}{h} = 1 - \epsilon.$$

On the other hand, with probability $\bar{P}_1 \geq \epsilon$ a bipartite subgraph B_i will end up not being optimized (i.e., the number of nodes of the subgraph in the final cover will be $V_2 = (1 - \epsilon)h$ or $V_2 = (1 - \epsilon)h + 1$).

Let a generation be *relevant* if it is decisive toward the optimization of a subgraph B_i (i.e., a generation where for the first time at least one bit is removed from a B_i subgraph). For the RLS algorithm there are exactly l *relevant* generations, one for each B_i subgraph.

Since, the v^* nodes will end up in the final cover anyway (i.e., they are in both the local and global covers), their selection is not influential for the optimization of a B_i subgraph. Apart for the v^* node, the bipartite subgraphs are *independent*. The probability that each is optimized is at most $1 - \epsilon$.

Let a relevant step be *successful* if after it happens the subgraph B_i will end up being optimized. Furthermore, let a relevant step be unsuccessful if it leads to the subgraph being not optimized. The probability that l *successful* iterations occur without any unsuccessful ones is less than

$$P_{opt} \leq (1-\epsilon)^l \leq (1-n^{\delta-1/2})^{\sqrt{n}}$$
$$= \left[(1-n^{\delta-1/2})^{n^{1/2-\delta}}\right]^{n^{\delta}} \leq e^{-n^{\delta}}$$

which is exponentially small. This leads to an expected number of at least $e^{n^{\delta}}$ different runs before the optimum is found. Since

RLS only flips one bit at a time, once the algorithm reaches a local optimum, it will be stuck forever. Hence, the expected optimization time is infinite. This proves the first part of the lemma. The second part of the proof follows.

Each bipartite subgraph has a probability of at most $(1-\epsilon)$ of being optimized. With a probability of exactly $(1-\epsilon)$, the expected number of optimized bipartite subgraphs would be at most $(1-\epsilon)l$ while at least ϵl would be the expected ones not to be optimized. Since the B_i subgraphs are independent, Chernoff bounds may be applied. By using Chernoff bounds it follows that the probability that the number of nonoptimized B_i's is less than $(1-n^{-\delta/3})\epsilon l$ is overwhelmingly small. In particular, let X be the random variable representing the number of nonoptimized bipartite subgraphs. Then

$$
\begin{aligned}
P\left(X \leq (1-n^{-\delta/3})\epsilon l\right)
&\leq \exp\left(-\frac{\epsilon l}{2n^{2\delta/3}}\right)\\
&\leq \exp\left(-\frac{n^{-1/2+\delta}n^{1/2}}{2n^{2\delta/3}}\right) = e^{-\Omega(n^{\frac{1}{3}\delta})}.
\end{aligned}
$$

So, with overwhelming probability the solution will have at least $(1-n^{-\delta/3})\epsilon l$ nonoptimized bipartite subgraphs with $(1-\epsilon)h+1$ nodes each in the cover, but one that will not have the v_i^* node in the cover (i.e., $(1-\epsilon)h$ nodes). On the other hand, the optimized bipartite graphs will be at most $l(1-\epsilon)+\epsilon l/n^{\delta/3}$ having $\epsilon h+1$ nodes in the cover each. This leads to an approximation of

$$
\begin{aligned}
&\frac{(1-n^{-\delta/3})\epsilon l[(1-\epsilon)h+1]+[l(1-\epsilon)[\epsilon h+1]]}{(\epsilon h+1)l}\\
&\quad+\frac{\frac{\epsilon l}{n^{\delta/3}}[\epsilon h+1]-1}{(\epsilon h+1)l}\\
&=(1-\epsilon)+\frac{[1-n^{-\delta/3}]\epsilon[(1-\epsilon)h+1]}{\epsilon h+1}+\frac{\epsilon}{n^{\delta/3}}\\
&\quad-\frac{1}{(\epsilon h+1)l}\\
&=(1-\epsilon)+(1-\epsilon)\frac{(1-n^{-\delta/3})\epsilon h}{\epsilon h+1}+\frac{(1-n^{-\delta/3})\epsilon}{\epsilon h+1}\\
&\quad+\frac{\epsilon}{n^{\delta/3}}-\frac{1}{(\epsilon h+1)l}\\
&=(1-\epsilon)+(1-\epsilon)\frac{(1-n^{-\delta/3})(\epsilon h+1)}{\epsilon h+1}\\
&\quad-(1-\epsilon)\frac{(1-n^{-\delta/3})}{\epsilon h+1}+\frac{(1-n^{-\delta/3})\epsilon}{\epsilon h+1}+\frac{\epsilon}{n^{\delta/3}}\\
&\quad-\frac{1}{(\epsilon h+1)l}\\
&=(1-\epsilon)+(1-n^{-\delta/3})(1-\epsilon)-\frac{(1-2\epsilon)(1-n^{-\delta/3})}{\epsilon h+1}\\
&\quad+\frac{\epsilon}{n^{\delta/3}}-\frac{1}{(\epsilon h+1)l}\\
&=2(1-\epsilon)-\frac{1-2\epsilon}{n^{\delta/3}}-\frac{(1-2\epsilon)(1-n^{-\delta/3})}{\epsilon h+1}-\frac{1}{(\epsilon h+1)l}\\
&=2(1-\epsilon)-o(1).
\end{aligned}
$$

This proves the theorem for the RLS algorithm starting with a *full* cover. ■

Lemma 18: Let $\epsilon > n^{-1/2+\delta}$, with $0<\delta<1/2$ a constant. The expected time for (1+1)-EA to find the minimum cover of the $B_{h,l}$ instance class is exponential in the number of nodes of the graph. With an overwhelming probability, the (1 + 1)-EA does not find an approximation that is better than $2(1-\epsilon)-o(1)$ in polynomial time.

Proof: The proof for the (1 + 1)-EA has to take into account the possibility that flips of higher number of bits may occur.

Let the algorithm be initialized with all the nodes in the cover. For each bipartite subgraph, the probability it is optimized when only single bit flips are considered is less than $(1-\epsilon)$ as proved for the RLS algorithm. At the beginning, higher number of bit flips of a bipartite subgraph B_i will only be accepted if they all belong to the V_1 subgraph or all belong to the V_2 subgraph.

The probability that a single bit flip is the first flip to act on a subgraph B_i is

$$P_{B(1\text{bit})}=\frac{b}{n}\left(1-\frac{1}{n}\right)^{n-1}\geq\frac{\sqrt{n}}{en}=\frac{1}{e\sqrt{n}}.$$

On the other hand, the probability that two bit flips or higher are the first to remove the nodes from a given subgraph B_i is at most

$$P_{B(\geq 2\text{bit})}\leq\binom{h+1}{2}\frac{1}{n^2}\leq\frac{n}{n^2}\leq\frac{1}{n}.$$

If a 2-bit flip that removes one bit from one subgraph B_i and one bit from a different subgraph (i.e., B_j with $j\neq i$) happens, then it is considered that two separate 1-bit flips have occurred. This is possible because of the independence of the bipartite subgraphs.

The bound above also holds for higher number of bits because in any case at least two bits have to flip. For more than two bits the probabilities are lower. The probability that two or more bit flips occur on a subgraph and are accepted before a one-bit flip is at most

$$\frac{1/n}{1/n+1/(e\sqrt{n})}\leq\frac{e\sqrt{n}}{n+e\sqrt{n}}\leq\frac{e}{\sqrt{n}}.$$

So with probability at most $\frac{e}{\sqrt{n}}$ the first nodes to be removed from a subgraph B_i are more than one. The probability that $\sqrt[3]{n}$ out of a total of $l=\sqrt{n}$ subgraphs are subject to this is

$$
\begin{aligned}
\binom{\sqrt{n}}{\sqrt[3]{n}}\left(\frac{e}{\sqrt{n}}\right)^{(n^{1/3})}&\leq\frac{\sqrt{n}^{n^{1/3}}}{(n^{1/3})!}\left(\frac{e}{\sqrt{n}}\right)^{n^{1/3}}\\
&=\frac{e^{n^{1/3}}}{(n^{1/3})!}\leq e^{-\Omega(\sqrt[3]{n})}. \qquad (2)
\end{aligned}
$$

So with overwhelming probability at most $\sqrt[3]{n}$ subgraphs are initially subject to more than one bit flip.

Assuming optimistically that the 2-bit flips or higher lead to a correct optimization of each subgraph B_i, the approximation factor found by the algorithm is at least

$$\left(1-\frac{\sqrt[3]{n}}{\sqrt{n}}\right)\left(2(1-\epsilon)-o(1)\right)=2(1-\epsilon)-o(1).$$

It obviously also follows that the global optimum will not be found with overwhelming probability before a *local search cover*. Once such a cover has been reached, the next accepted mutation requires at least $2V_1$ nodes to flip. The expected time for a $2V_1$-bit flip is exponential in the problem size. This proves the exponential expected time for the global optimum to be reached.

The above arguments only consider the first bits removed from either the V_1 or the V_2 subgraphs of each B_i. There is also a probability that flips of two or more bits may invert the direction of the process (i.e., an inversion occurs). In particular, if a V_2 node of a B_i is removed before a V_1 node of the same bipartite subgraph, then a 2-bit flip may invert the direction of the process leading to a better local optimum than the one proved for the RLS algorithm.

Let an *inversion toward the optimum* of a B_i subgraph occur if all the V_1 nodes of the B_i missing in the cover are inserted in the cover and at least one V_2 node of the B_i is removed. An *inversion away from the optimum* is the opposite (i.e., all the missing V_2 nodes are inserted in the cover and at least one V_1 node is removed from the cover).

The probability of an inversion toward the optimum of a subgraph is highest when only one V_1 node is missing from the cover. In this case it is sufficient that it is flipped together with one V_2 node. Hence, the probability is

$$p_{\text{inv}(1)} \leq (1-\epsilon)h\frac{1}{n^2} = \frac{(1-\epsilon)}{n\sqrt{n}}.$$

On the other hand, the probability another V_1 node is removed is at least $1/(en)$. So the probability an inversion of a B_i occurs before another V_1 node is removed from the B_i is less than

$$\frac{1/(n\sqrt{n})}{1/(n\sqrt{n}) + 1/(en)} = \frac{e}{\sqrt{n}+e} \leq \frac{e}{\sqrt{n}}.$$

If the inversion above does not occur before another V_1 node is removed, then the probability that an inversion occurs afterward is lower, because at least two V_1 nodes have to be flipped together with at least one V_2 node.

Let $P_{\text{inv}(i)}$ be the probability of an inversion when i V_1 nodes are missing from the cover. Then

$$P_{\text{inv}(i)} \leq (1-\epsilon)h\frac{1}{n^{i+1}} = (1-\epsilon)\frac{1}{\sqrt{n}n^i} \leq \frac{1}{\sqrt{n}n^i}.$$

Since the probability an extra V_1 node is removed is at least $1/(en)$, the probability an inversion happens before is less than

$$\frac{1/(\sqrt{n}n^i)}{1/(\sqrt{n}n^i) + 1/(en)} = \frac{e}{\sqrt{n}n^{i-1}+e} \leq \frac{e}{\sqrt{n}n^{i-1}}.$$

This gives an inversion probability for a subgraph of not more than

$$\sum_{i=1}^{\epsilon h+1}\frac{e}{\sqrt{n}n^{i-1}} \leq \frac{e}{\sqrt{n}}\sum_{i=1}^{\epsilon h+1}\frac{1}{n^{i-1}} \leq \frac{e}{\sqrt{n}}\sum_{i=0}^{\epsilon h+1}\frac{1}{n^i} \leq \frac{e^2}{\sqrt{n}}.$$

As done previously [i.e., (2)], the probability that not more than a very small amount (for example $\sqrt[3]{n}$) subgraphs are subject to these inversions can be proved to be exponentially high.

The above argument shows that with overwhelming probability also the (1 + 1)-EA does not produce a better approximation than that calculated for the RLS algorithm. ■

Theorem 12 follows from Lemmas 17 and 18. The theorem has only been proved for a full cover set initialization. The previous analyzes presented in this paper have given an idea that the initialization is not important for the covering capabilities of the RLS algorithm and the (1 + 1)-EA, at least for graphs with regular structures as those considered in this paper. We believe that similar results can be obtained for the other initializations considered in this paper.

The approximation ratio proved in the theorem is less than two, but tends to it. Since the result holds with an overwhelming probability, a restart strategy will not be effective for producing better approximations. The state-of-the-art algorithm for vertex cover has a worst case approximation ratio that tends to two. So the theorem proves that even by using multiple runs the RLS algorithm and (1 + 1)-EA are not better worst case approximation algorithms for vertex cover compared to the state-of-the-art problem-specific approximation algorithm.

VIII. Conclusion

A computational complexity analysis of the RLS algorithm and of the (1 + 1)-EA for the vertex cover problem has been presented. From the analysis of different instance classes, characteristics of graphs that make the optimization task easy or hard for the algorithms have been highlighted.

If the degree of each node in a graph is not higher than 2, then the (1 + 1)-EA can find the minimum cover efficiently. Such a result is proved in Section VI. The reason for this efficiency is that if each node of the graph is connected with at most two other nodes, then the algorithm will be able to escape from any local optima by flipping at most two bits. If the number of nodes in the graph is large enough, it is very likely that the algorithm will have to flip two bits at a time to find the minimum cover. Since the RLS algorithm only flips one bit per iteration, it is not efficient on all graphs with vertex degree at most two and requires infinite expected optimization time because once it reaches a local optimum it will be stuck forever. The RLS algorithm may be made efficient for the problem if the neighborhood size of its mutation operator is changed. In particular, if the algorithm is allowed to flip either one bit or two bits in each iteration, then it will also be efficient for any graph with vertex degree at most two.

However, as the vertex degree increases, also the (1 + 1)-EA is challenged by a harder optimization task. Differently compared to the RLS algorithm, as long as the vertex degree of the graph is constant (i.e., does not depend on the number of nodes in the graph), the (1 + 1)-EA can escape from local optima efficiently because at most a constant number of bits need to be flipped and this requires polynomial time. This is not true for the RLS algorithm, which will be stuck forever on a local optimum because it only flips one bit in each iteration.

The (1 + 1)-EA encounters greater difficulties if the vertex degree of some nodes depends on the problem size (i.e., it depends on the number of nodes in the graph). As it can be derived from the analysis of the PS (Section III) or of the bipartite (Section IV) instance classes, the runtime required by the (1 + 1)-EA to escape from a local optimum may be exponential in the problem size with high probability (i.e., a constant probability or an inversely polynomial probability), because the time required to contemporaneously flip a number of bits, which depends on the number of nodes in the graph, is exponential. This implies that the expected runtime of the algorithm is also exponential in the problem size. Furthermore, as seen for the bipartite graph, the local optimum on which the algorithms may get trapped could have a cover value that is considerably worse than to that of the minimum cover. It is this problem that leads to the very bad worst case approximation ratios of the algorithms.

As long as the number of these "traps" (i.e., vertex subclasses with a high vertex degree) is constant in the problem size, as for the previously considered instance classes, both algorithms may overcome the problem by using restart strategies. In other words, the algorithm has a high enough probability of ending either on the global optimum or on a local optimum. If the local optimum is found first, then exponential time is required to escape from the "trap." However, since the probability of reaching the global optimum is high (i.e., constant or polynomial), then, if the algorithm is run a sufficient number of times, it will find the global optimum efficiently in at least one of these runs. Interestingly, if there is a high enough probability that the global optimum is found before reaching a "trap" by just flipping one bit at a time, then also the RLS algorithm will be efficient if a restart strategy is used. In fact, as proved in Sections III and IV, this is the case for the PS and the bipartite instance classes.

Matters change if the number of such "traps" depends on the size of the problem. If there are too many "traps," a restart strategy may also be inefficient because an exponential number of restarts may be necessary to guarantee that at least in one run the algorithms do not get trapped in a local optimum. In other words, the RLS algorithm and the (1 + 1)-EA will not find the minimum cover of the graph efficiently even if they are run many times, because the probability that all the traps are avoided is very low. The only way around the problem is to accept suboptimal solutions which can be found quickly.

As shown in Sections V and VII, if there are many of such traps, then with a high probability the algorithms avoid quite a few of them and reasonable approximations may be found. Rather than searching exclusively for the global optimum, solutions that differ from the optimum of at most a certain factor (i.e., the approximation ratio) are also accepted. Concerning the $G_{h,l}$ instance class of Section V, an approximation of $4/3$ can be guaranteed by both the RLS algorithm and the (1 + 1)-EA in $O(n \log n)$ time. Hence, accepting suboptimal solutions may reduce the runtime from exponential to subquadratic.

Nevertheless, in Section VII it is proved that neither algorithm can guarantee a better approximation ratio than that of the best problem specific algorithm for vertex cover.

The algorithms considered in this paper only use population size 1. The analysis of these algorithms has greatly contributed to the understanding of what kinds of landscapes of the vertex cover problem are easy and which are hard for single individuals evolving through a mutation-based process. However, the EAs used in practice make use of a population of individuals and of other perturbation operators, such as crossover, rather than counting only on mutation. Hopefully, the presented analysis will be helpful in starting to understand on which kind of vertex cover landscapes a population or a different operator may be useful. As a natural extension to the work presented in this paper, we plan to analyze population and crossover-based algorithms for vertex cover to understand whether these algorithms may overcome the problems of the (1+1)-EA and what kind of other problems they may encounter.

References

[1] P. S. Oliveto, J. He, and X. Yao, "Evolutionary algorithms and the vertex cover problem," in *Proc. Congr. Evol. Comput. (CEC '07)*, Singapore, Sep. 2007, pp. 1870–1877.

[2] R. Sarker, M. Mohammadian, and X. Yao, *Evolutionary Optimization*. Norwell, MA: Kluwer, 2002.

[3] G. Rudolph, "Finite Markov chain results in evolutionary computation: A tour d'horizon," *Fundamenta Informaticae*, vol. 35, no. 1–4, pp. 67–89, 1998.

[4] S. Droste, T. Jansen, and I. Wegener, "On the optimization of unimodal functions with the (1+1) evolutionary algorithm," in *Proc. 5th Int. Conf. Parallel Problem Solving from Nature*, London, U.K.: Springer-Verlag, 1998, pp. 13–22.

[5] T. Jansen and I. Wegener, "Evolutionary algorithms: How to cope with plateaus of constant fitness and when to reject strings of the same fitness," *IEEE Trans. Evol. Comput.*, vol. 5, no. 6, pp. 589–599, Dec. 2001.

[6] I. Wegener, "Methods for the analysis of evolutionary algorithms on pseudo-boolean functions," in *Proc. Evol. Optimization*, Dordrecht, The Netherlands: Kluwer, 2001, pp. 349–369.

[7] J. He and X. Yao, "Toward an analytic framework for analysing the computation time of evolutionary algorithms," *Artif. Intell.*, vol. 145, no. 1–2, pp. 59–97, 2003.

[8] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1 + 1) evolutionary algorithm," *Theor. Comput. Sci.*, vol. 276, no. 1–2, pp. 51–81, 2002.

[9] I. Wegener and C. Witt, "On the analysis of a simple evolutionary algorithm on quadratic pseudo-boolean functions." *J. Discrete Algorithms*, vol. 3, no. 1, pp. 61–78, 2005.

[10] O. Giel and I. Wegener, "Evolutionary algorithms and the maximum matching problem," in *Proc. 20th Annu. Symp. Theoretical Aspects Comput. Sci. (STACS '03)*, London, U.K.: Springer-Verlag, pp. 415–426.

[11] J. He and X. Yao, "Time complexity analysis of an evolutionary algorithm for finding nearly maximum cardinality matching," *J. Comput. Sci. Technol.*, vol. 19, no. 4, pp. 450–458, 2004.

[12] F. Neumann and I. Wegener, "Randomized local search, evolutionary algorithms, and the minimum spanning tree problem," *Theoretical Comput. Sci.*, vol. 378, no. 1, pp. 32–40, 2007.

[13] C. Witt, "Worst-case and average-case approximations by simple randomized search heuristics," in *Proc. 22nd Annu. Symp. Theor. Aspects Comput. Sci. (STACS '05)*, LNCS vol. 3404, pp. 44–56.

[14] J. He and X. Yao, "Drift analysis and average time complexity of evolutionary algorithms," *Artificial Intell.*, vol. 127, no. 1, pp. 57–85, 2001.

[15] J. He and X. Yao, "A study of drift analysis for estimating computation time of evolutionary algorithms," *Natural Computing: Int. J.*, vol. 3, no. 1, pp. 21–35, 2004.

[16] P. S. Oliveto, J. He, and X. Yao, "Time complexity of evolutionary algorithms for combinatorial optimization: a decade of results," *Int. J. Automation Computing*, vol. 4, no. 3, pp. 281–293, 2007.

[17] S. Khuri and T. Bäck, "An evolutionary heuristic for the minimum vertex cover problem," in *Proc. KI-94 Workshop Genetic Algorithms Within Framework Evol. Comput.*, Saarbrücken, Germany, 1994, pp. 86–90.

[18] G. H. Sasaki and B. Hajek, "The time complexity of maximum matching by simulated annealing," *J. ACM*, vol. 35, no. 2, pp. 387–403, 1988.

[19] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 1995.

[20] T. Storch, "Finding large cliques in sparse semi-random graphs by simple randomised search heuristics," *Theoretical Comput. Sci.*, vol. 386, no. 1–2, pp. 114–131, 2007.

[21] E. Halperin, "Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs," in *Proc. Symp. Discrete Algorithms*, 2000, pp. 329–337.

[22] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. New York: W. H. Freeman, 1979, ch. 4, sec. 1, p. 84.

[23] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. New York: Dover, 1998, ch. 15, sec. 6, pp. 360–363.

[24] J. He, X. Yao, and J. Li, "A comparative study of three evolutionary algorithms incorporating different amounts of domain knowledge for node covering problems," *IEEE Trans. Syst., Man, Cybern., Part C*, vol. 35, no. 2, pp. 266–271, 2005.

[25] T. H. Cormen, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. 2nd ed. New York: McGraw-Hill, 2001, ch. 3, sec. 1, pp. 41–50.

[26] I. K. Evans, "Evolutionary algorithms for vertex cover," in *Proc. Evol. Programming VII*, Berlin, Germany: Springer-Verlag, 1998, pp. 377–386.

[27] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt, "Approximating covering problems by randomized search heuristics using multiobjective models," in *Proc. Genetic Evol. Comput. Conf. (GECCO '07)*, London, U.K., Jul. 2007, pp. 797–804.

[28] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt, "On improving approximate solutions by evolutionary algorithms," in *Proc. Congr. Evol. Comput. (CEC '07)*, Singapore, Sep. 2007, pp. 2614–2621.

[29] B. Hajek, "Hitting-time and occupation-time bounds implied by drift analysis with applications," *Advances Appl. Probability*, vol. 14, no. 3, pp. 502–525, 1982.

[30] P. S. Oliveto and C. Witt, "Simplified drift analysis for proving lower bounds in evolutionary computation," in *Proc. 10th Int. Conf. Parallel Problem Solving Nature*, 2008, pp. 82–91.

[31] N. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory*. New York: Oxford Univ. Press, 1976.

[32] S. Droste, T. Jansen, and I. Wegener, "A rigorous complexity analysis of the $(1+1)$ evolutionary algorithm for separable functions with boolean inputs." *Evol. Comput.*, vol. 6, no. 2, pp. 185–196, 1998.

[33] J. Clark and D. A. Holton, *A First Look at Graph Theory*. Singapore: World Scientific, 1991.

Pietro S. Oliveto (S'07) received the Laurea degree in computer science from the University of Catania, Catania, Italy, in 2005 and the Ph.D. degree in computer science from the University of Birmingham, Birmingham, U.K., in 2009, with a dissertation on the computational complexity analysis of evolutionary algorithms.

Currently, he is an EPSRC funded research fellow at the School of Computer Science, University of Birmingham. His main research interest is the time complexity analysis of randomized algorithms for combinatorial optimization problems. He is currently considering evolutionary, ant colony and artificial immune system algorithms.

Jun He (M'06) received the Ph.D. degree in computer science from Wuhan University, Wuhan, China, in 1995.

He was a research fellow in the School of Computer Science, University of Birmingham, Birmingham, U.K., between 2001 and 2007. Currently, he is with the Department of Computer Science, Aberystwyth University, Aberystwyth, Wales, U.K. His research interests include evolutionary computation, data mining and network security.

Xin Yao (M'91–SM'96–F'03) obtained the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the M.Sc. degree from the North China Institute of Computing Technology, Beijing, China, in 1985, and the Ph.D. degree from USTC in Hefei, China, in 1990.

He was an Associate Lecturer and Lecturer between 1985 and 1990 at USTC. He was a Postdoctoral Fellow in the Computer Sciences Laboratory at the Australian National University in Canberra in 1990, and continued his work on simulated annealing and evolutionary algorithms. He joined the Knowledge-Based Systems Group at Commonwealth Scientific and Industrial Research Organization, Division of Building, Construction, and Engineering in Melbourne in 1991, working primarily on an industrial project on automatic inspection of sewage pipes. He returned to Canberra in 1992 to take up a lectureship with the Australian Defense Force Academy, in the School of Computer Science, University of New South Wales, where he was later promoted to Senior Lecturer and Associate Professor. He moved to the University of Birmingham, England, as a Professor (Chair) of Computer Science in 1999. Currently, he is the Director of the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham, U.K. He is also a Distinguished Visiting Professor of the University of Science and Technology of China, Hefei, and a Visiting Professor at three other universities. He has more than 200 refereed research publications. He serves as Associate Editor or Editorial Board Member of several journals, and is the Editor of the book series "Advances in Natural Computation (World Scientific)." He has been invited to give more than 45 invited keynote and plenary speeches at conferences and workshops worldwide. His major research interests include evolutionary computation, neural network ensembles, and their applications.

Prof. Yao is a Distinguished Lecturer of IEEE Computational Intelligence Society. He was the Editor-in-Chief from 2003 to 2008 of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION. He won the 2001 IEEE Donald G. Fink Prize Paper Award for his work on evolutionary artificial neural networks. His doctoral work on simulated annealing and evolutionary algorithms earned him the President's Award for Outstanding Thesis by the Chinese Academy of Sciences.