Analytical and experimental comparison of six algorithms for the vertex cover problem

Article *in* ACM Journal of Experimental Algorithmics · March 2010

DOI: 10.1145/1865970.1865971

2 authors, including:

Francois Delbot
University Paris Nanterre
41 PUBLICATIONS **297** CITATIONS

Analytical and experimental comparison of six algorithms for the vertex cover problem

FRANÇOIS DELBOT
IBISC, Université d'Evry, France
and
CHRISTIAN LAFOREST
LIMOS, Université Blaise Pascal, France

The *vertex cover* is a well-known NP-complete minimization problem in graphs that has received a lot of attention these last decades. Many algorithms have been proposed to construct vertex cover in different contexts (offline, online, list algorithms, etc.) leading to solutions of different level of quality. This quality is traditionally measured in terms of *approximation ratio*, that is the worst possible ratio between the quality of the solution constructed and the optimal one. For the vertex cover problem the range of such known ratios are between 2 (conjectured as being the smallest constant ratio) and Δ, the maximum degree of the graph. Based on this measure of quality, the hierarchy is almost clear (the smaller the ratio is, the better the algorithm is).

In this paper, we show that this measure, although of great importance, is too macroscopic and does not reflect the practical behavior of the methods. We prove this by analyzing (known and recent) algorithms running on a particular class of graphs: the paths. We obtain closed and exact formulas for the mean of the sizes of vertex cover constructed by these different algorithms. Then, we assess their quality experimentally in several well-chosen class of graphs (random, regular, trees, BHOSLIB benchmarks, trap graphs, etc.). The synthesis of all these results lead us to formulate a "practical hierarchy" of the algorithms. We remark that it is, more or less, the opposite to the one only based on approximation ratios, showing that worst case analysis only gives partial information on the quality of an algorithm.

Categories and Subject Descriptors: F.2 [**THEORY OF COMPUTATION**]: ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY; G.2.2 [**DISCRETE MATHEMATICS**]: Graph Theory

General Terms: Algorithms, Experimentation, Performance, Theory

Additional Key Words and Phrases: differential approximation, mean analysis, vertex cover, worst case approximation

Author's address: François Delbot, Laboratoire IBISC, FRE CNRS 3190, Université d'Evry, Tour Evry 2, 523 place des terrasses, 91000 Evry, France; email: francois.delbot@ibisc.fr
Christian Laforest, LIMOS, UMR 6158 CNRS, Université Blaise Pascal, Campus scientifique des Cézeaux, 24, Avenue des Landais, BP 10 125, 63173 Aubiere cedex, France; email: christian.laforest@isima.fr

This work is partially supported by the projet ToDo (Time versus Optimality in Discrete Optimization) funded by the French ANR and by the project "Approximation Rapide" funded by the French GDR RO

1. INTRODUCTION

Graph theory is widely used to model many different types of concrete situations (logistic, networks, social science, etc.). The resulting graphs are then manipulated by algorithms, often to optimize a given objective. Unfortunately, many discrete optimization problems are intractable, *i.e.* cannot be solved exactly in polynomial time. This is the case for the numerous NP-complete problems (see [Garey and Johnson 1979]). However, due to their theoretical and practical importance they must be solved. Many alternatives are possible. The first one is running heuristics; in most cases, there is no guarantee neither on the running time nor on the quality of the solution produced. Another approach is (sometimes) possible, namely the *approximation algorithms*; such an algorithm must have a polynomial time complexity **and** must return a solution those quality is in a given range between optimal, noted OPT, and $r \cdot OPT$ where $r \geq 1$ is the *approximation ratio* (this terminology is for *minimization* problem). If such an approximation algorithm exists, we have two guarantees: the running time is "reasonable" *and* the quality of the solution is in a range, given by the *analysis* of the algorithm (the approximation ratio must be analytically *proved*). Approximation algorithms are then interesting tools to produce solutions with proven quality to discrete optimization problems a lot of works have been devoted to this subject [Hochbaum 1997; Ausiello et al. 1999; Vazirani 2002]. In such situations, the "quality" of such an algorithm is measured by its approximation ratio r (that is not necessarily a constant). If r is small the quality is good.

This paper is devoted to the comparison of several approximation algorithms for a well known minimization problem, namely the *vertex cover* problem. Let $G = (V, E)$ be any undirected non weighted graph where V is the set of *vertices* and E the set of *edges*. A *vertex cover* C is a subset of the vertices ($C \subseteq V$) such that each edge uv has at least one extremity in C ($u \in C$ or $v \in C$ or both). The associated optimization problem is to construct a vertex cover of minimum size. This is a classical optimization problem that has several approximation algorithms. The vertex cover problem is often involved in the resolution of other problems (most often at an intermediate stage) such as multiple sequence alignment [Roth-Korostensky 2000], conflict resolution in sequence data [Stege 2000] or network monitoring [Zhang et al. 2006].

The best constant approximation ratio known is 2 and it is conjectured (see for example [Khot and Regev 2008]) that there is no smaller *constant* ratio. Hence, 2 seems to be the "best" possible result reachable. Several algorithms have this ratio (and are described and studied in this paper) and are known and are used since many years.

Online and list algorithms. More recently other "contexts" of applications have emerged: for example, in an *online* context, the vertices are *revealed* one by one (with their edges incident to already revealed vertices) and a decision to include or not the current vertex must be irrevocably taken during this step. In the same spirit, *list algorithms* have been investigated. In this situation the graph is known but presented to the algorithm piece by piece following a given order (the list).
In these works, the intrinsic hardness of the problem is "complicated" by the running context where decisions must be taken based on partial information. Then,

it is not surprising that the algorithms designed to produce a vertex cover in such contexts have bad (and often non constant) approximation ratios.

Thus, several (polynomial time) algorithms have been proposed for the vertex cover problem with approximation ratios lying between 2 and Δ (the maximum degree of the graph). The hierarchy based on this criterium is rather clear and the considered best algorithms are those who have a ratio of 2.

However, in practice, the situation is not so clear. This is what we show in this paper. The main goal of our work is to establish a hierarchy of these algorithms, based on practical behavior. Indeed, even if they are deterministic, the final result is influenced by "local choices" made during the construction. When such a choice is possible, we consider that it is made randomly and uniformly. We can then study these algorithms not in the *worst case* scenario (as for the approximation ratio) but in an *average* case. This is closer to the real behavior of the method since worst cases are often rare and artificial and do not represent the range of all possibilities. Our main result is that the "practical" hierarchy based on average study we obtain at the end of our study is more or less the opposite of the one based on the approximation ratio.

The choice of the algorithms studied. Our choice was mainly based on the two following criterium:

—One of our future objective is to adapt some of them to the treatment of very large instances. With this perspective in mind we are only interested by algorithms with low complexity and involving basic mechanisms.

—The comparison must be done "fairly". This can only be done if all the steps of computation are completely described and specified. This is why we did not consider here algorithms involving linear programing for example since the precision and the performances may depend on the solver used.

Outline of the paper. In Section 2 we present the algorithms that we study and their approximation ratio. In Section 3 we make an exact mean analysis of their performance on a particular class of graphs (the *path graphs*). Then in Section 4 we conduct an experimental study on several classical graphs classes. We make a conclusion in Section 5.

2. PRESENTATION OF THE ALGORITHMS

In this section, we present six algorithms for the vertex cover problem, namely Maximum Degree Greedy, Greedy Independent Cover, Depth First Search, Edge Deletion, ListLeft and ListRight.

Remark. In the following descriptions, when an algorithm removes one or more vertices, incident edges are also deleted. Then, the set E (containing all edges of the graph) is updated implicitly.

2.1 The algorithms

The Maximum Degree Greedy (MDG) Algorithm is an adaptation of the classical greedy algorithm for the set cover problem (see for example [Cormen et al. 2009]). It's worst-case approximation ratio is $H(\Delta)$, with $H(n) = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$

the harmonic series ($H(n) \approx \ln n + 0.57$ when n is large) and Δ the maximum degree of the graph.

Algorithm 1: Maximum Degree Greedy (MDG)

```
Data: a graph G = (V, E)
Result: a vertex cover of G
C ← ∅;
while E ≠ ∅ do
    select a vertex u of maximum degree;
    V ← V − {u};
    C ← C ∪ {u};
end
return C;
```

The Greedy Independent Cover (GIC) Algorithm is an adaptation of the well know greedy algorithm presented in [Halldórsson and Radhakrishnan 1994] for the independent set problem. It's worst-case approximation ratio is at least of $\frac{\sqrt{\Delta}}{2}$ (this lower bound is obtained with graphs presented in [Avis and Imamura 2007]). We note $N(u)$ the set of neighbors of the vertex u.

Algorithm 2: Greedy Independent Cover (GIC)

```
Data: a graph G = (V, E)
Result: a vertex cover of G
C ← ∅;
while E ≠ ∅ do
    select a vertex u of minimum degree;
    C ← C ∪ N(u);
    V ← V − (N(u) ∪ {u});
end
return C;
```

The Depth First Search (DFS) algorithm has worst-case approximation ratio of 2 and was presented in [Savage 1982]. This algorithm returns the nonleaf vertices of a depth-first search spanning tree. Hence, if G is connected the result is a connected vertex cover. Otherwise the algorithm must be executed on each connected component of G (excepted isolated vertices). To simplify the description we suppose that G is connected.

Algorithm 3: Depth First Search (DFS)

```
Data: a (connected) graph G
Result: a vertex cover of G
Compute T, a depth-first search spanning tree of G starting from any vertex r.
Let I(T) be the set of nonleaves vertices of T.
return I(T) ∪ {r};
```

The Edge Deletion (ED) Algorithm, proposed by Gavril (see [Garey and Johnson 1979]), returns the vertices of a maximal (but not necessarily maximum) matching. It's worst-case approximation ratio is 2 (see for example [Cormen et al. 2009]). More precisely, the tight worst-case approximation ratio is asymptotic to $\min\{2, \frac{1}{1-\sqrt{1-\epsilon}}\}$ for graphs with an average degree of at least ϵn and to $\min\{2, \frac{1}{\epsilon}\}$ for graphs with a minimum degree of at least ϵn (see [Cardinal et al. 2005]).

Algorithm 4: Edge Deletion (ED)

Data: a graph $G = (V, E)$
Result: a vertex cover of G
$C \leftarrow \emptyset$;
while $E \neq \emptyset$ **do**
  select $uv \in E$;
  $C \leftarrow C \cup \{u, v\}$;
  $V \leftarrow V - \{u, v\}$;
end
return C;

The ListLeft (LL) Algorithm, proposed by Avis and Imamura in [Avis and Imamura 2007], is a *list heuristic*. In this model, an algorithm scans the vertices one by one in a fixed given order (called a *list*) and takes a decision for each currently scanned vertex (and each decision is definitive). This can be viewed as an intermediate model between traditional approximation algorithms or heuristics (where the vertices can be treated dynamically in any order at each step of computation) and the pure on-line model studied in [Demange and Paschos 2005] (where the order of revelation is imposed by an adversary and each decision is definitive). In list heuristics, the graph and the order of scan (i.e. the list) are known in advance but *cannot be updated* during the process. For a given list $\mathcal{L} = u_1, \ldots, u_n$ we say that u_i is *on the left* (resp. *on the right*) of u_j if $i < j$ (resp. $i > j$); in addition, if u_i and u_j are neighbors in the graph then u_i is called a *left neighbor* (resp. *right neighbor*) of u_j.

In [Avis and Imamura 2007], the authors show that the ListLeft Algorithm has an approximation ratio of $\frac{\sqrt{\Delta}}{2} + \frac{3}{2}$ when lists are sorted by decreasing order of their degrees and that any list algorithm cannot have an approximation better than $\frac{\sqrt{\Delta}}{2}$ in that case.

Algorithm 5: ListLeft (LL)

Data: a graph G and an associated list $\mathcal{L}$
Result: a vertex cover of G
$C \leftarrow \emptyset$;
Scan the list $\mathcal{L}$ *from left* to right. Let u be the current scanned vertex.
if *u has at least a right neighbor not in C* **then**
  $C \leftarrow C \cup \{u\}$
end
return C;

The ListRight (LR) Algorithm is a better list heuristic than ListLeft; more precisely, it was proved in [Delbot and Laforest 2008] that for *any list* $\mathcal{L}$ ListRight returns a vertex cover whose size is smaller than or equal to the one constructed by ListLeft applied on the same list $\mathcal{L}$. ListRight has a worst-case approximation ratio of Δ (this bound is tight in stars).

Algorithm 6: ListRight (LR)

Data: a graph G and an associated list $\mathcal{L}$
Result: a vertex cover of G
$C \leftarrow \emptyset$;
Scan the list $\mathcal{L}$ *from right* to left. Let u be the current scanned vertex.
if *u has at least a right neighbor not in C* **then**
  $C \leftarrow C \cup \{u\}$
end
return C;

2.2 Probabilistic model

The worst case analysis cannot capture the whole behavior of an algorithm on different classes of graphs. The worst case analysis is a very useful tool in the analysis of algorithms, but it is not sufficient to determine *a priori* the performance of an algorithm on a specific class of graphs. For example, an algorithm may be forced to always return its worst case (see the Edge Deletion algorithm who always returns 2 vertices of a star with $n \geq 2$ vertices) while another may be forced to always return the best solution (see Maximum Degree Greedy which returns only the central vertex). The worst case approximation ratio is only a macroscopic view of worst case performance of an algorithm. For example, the approximation ratio in worst case of Maximum Degree Greedy is $H(\Delta) = \sum_{i=1}^{\Delta} \frac{1}{i}$, while its worst case approximation ratio on star with Δ leaves is 1. Moreover, many algorithms contains a part of non-determinism (e.g. in the choice of selecting the edge for Edge Deletion, or the choice of a vertex of maximum degree for Maximum Degree Greedy) which can strongly influence their performance. For example, when applying ListRight on a star with n vertices (eg a graph with $n-1$ vertices of degree 1 and 1 vertex of degree $n-1$) it is easy to show that it returns the optimal solution (the vertex of degree $n-1$) if the central vertex is not at the right end of the list, which is the case with probability $\frac{n-1}{n}$, showing that this algorithm is optimal with high probability and that its worst case corresponds to a rare execution.

One cannot understand the behavior of approximation algorithms only focusing on the worst case approximation ratio and it is necessary to use additional methods of evaluation, such as average evaluation for example.

In order to perform an average evaluation, we need to specify the probabilistic model used. In the rest of this paper, each time a choice will be made, it will be done equiprobably among all possible choices. In practice, these choices will

influence our algorithms as follows:

—The choose made on line 3 in the description of Maximum Degree Greedy, Greedy Independent Cover and Edge Deletion is made equiprobably. For example, Maximum Degree Greedy selects a vertex of maximum degree u equiprobably among the set of vertices of maximum degree.

—The first line of Depth First Search is dedicated to the construction of a depth first search spanning tree. We assume that a vertex is chosen to be the root with probability $\frac{1}{n}$ and that each time the algorithm explores a branch starting from a vertex u by selecting a neighbor v of u, v is chosen equiprobably among the neighbors of u.

—ListLeft and ListRight are deterministic but accept a list $\mathcal{L}$ as input data. We assume that $\mathcal{L}$ is chosen equiprobably among the $n!$ possible lists.

3. AVERAGE BEHAVIOR OF THESE ALGORITHMS ON PATHS

In this section, we study the exact behavior of different algorithms on a simple but nontrivial class of graphs: the paths. A path of size n (noted P_n in this paper) is a tree[1] with all vertices of degree 2 except two that are of degree 1. Let $Opt(G)$ be the size of an optimal vertex cover on G (note that $Opt(P_n) = \lfloor \frac{n}{2} \rfloor$). We give a closed formula of the expectation of the size of solutions returned by each algorithm.

3.1 Expectation of the size of solutions returned by each algorithm

Theorem 1. $\mathbb{E}\left(\text{Greedy Independent Cover}\,(P_n)\right) = Opt(P_n)$.

Proof. It is mentioned in [Halldórsson and Radhakrishnan 1994] that Greedy Independent Cover finds an optimal vertex cover in trees and therefore in paths. □

Theorem 2. $\mathbb{E}(\text{Depth First Search}(P_n)) = n - 2 + \frac{2}{n}$.

Proof. This algorithm constructs a depth-first search spanning tree rooted in a vertex r. With our equiprobability assumption, each vertex of P_n can be the root with probability $\frac{1}{n}$. If r is a vertex of degree 1 Depth First Search return a solution of size $n-1$. At the opposite, if r is a vertex of degree 2, the tree contains exactly 2 leaves (the two vertices of degree 1) and Depth First Search returns a solution of size $n-2$. Then, we can deduce the mean of the algorithm when applied to P_n: $\mathbb{E}(\text{Depth First Search}(P_n)) = \frac{2}{n}(n-1) + \frac{n-2}{n}(n-2) = n - 2 + \frac{2}{n}$. □

Theorem 3. $\mathbb{E}(\text{ListLeft}(P_n)) = \frac{2n-1}{3}$.

Proof. The mean of ListLeft on a graph G with n vertices is obtained by the following formula: $\mathbb{E}(\text{ListLeft}(G)) = n - \sum_{u \in V} \frac{1}{d(u)+1}$ (Eric Angel, personal communication). To see that, it suffices to note that a vertex u is not selected by ListLeft only if all its neighborhood is on the left in the list. By considering all the lists that meet this condition, we obtain that the probability of such an event is $\frac{1}{d(u)+1}$. Therefore, the probability of selecting a vertex u is $1 - \frac{1}{d(u)+1}$ and we obtain the claimed formula. In a path of size n, there is exactly $n-2$ vertices of

[1] A tree is a connected graph with n vertices and $n-1$ edges

degree 2 and two vertices of degree 1. by applying the formula to P_n, we obtain that $\mathbb{E}(\textsc{ListLeft}(P_n)) = n - \frac{1}{3}(n-2) - \frac{1}{2}2 = \frac{2n-1}{3}$. □

The following result has been shown in [Birmelé et al. 2009].

THEOREM 4. $\mathbb{E}(\textsc{ListRight}(P_n)) = \frac{n-1}{2} + \frac{n+1}{2} \cdot \sum_{k=0}^{n} \frac{(-2)^k}{k!} + \sum_{k=0}^{n-1} \frac{(-2)^k}{k!}$.

PROOF. See [Birmelé et al. 2009]. □

By applying similar techniques used in [Birmelé et al. 2009], we can show the expectation of MAXIMUM DEGREE GREEDY and EDGE DELETION. We only present the proof for MAXIMUM DEGREE GREEDY here.

THEOREM 5. $\mathbb{E}(\textsc{Maximum Degree Greedy}(P_n)) = \frac{n-1}{2} + \frac{n-1}{2} \sum_{p=0}^{n-1} \frac{(-2)^p}{p!} + \sum_{p=0}^{n-2} \frac{(-2)^p}{p!}$.

PROOF. The proof follows directly from Lemmas 6 and 7 that are proved in the following. □

LEMMA 6. *A recursive expression of* $\mathbb{E}(MDG(P_n))$ *is:*

$$\mathbb{E}(MDG(P_0)) = \mathbb{E}(MDG(P_1)) = 0 \text{ and } \mathbb{E}(MDG(P_2)) = 1$$

$$\mathbb{E}(MDG(P_n)) = 1 + \frac{2}{n-2} \sum_{k=2}^{n-2} \mathbb{E}(MDG(P_k))$$

PROOF. MDG selects a vertex if and only if there is at least one edge in the graph and therefore at least two vertices. Then, we deduce that $\mathbb{E}(MDG(P_0)) = \mathbb{E}(MDG(P_1)) = 0$.

The vertices of P_n are labeled from v_1 to v_n. At each step of the algorithm, MDG selects a vertex v_k of maximum degree. There are two cases:

—The path contains two vertices. MDG selects one of the two vertices and will not select the other vertex because the graph does not contain any edge. We obtain that $\mathbb{E}(MDG(P_2)) = 1$.

—The path contains more than two vertices. MDG selects a vertex of degree 2, that we call v_k, among the $n-2$ vertices of degree 2, with respect to the equiprobability assumption. After this step, we obtain two paths resulting from the deletion of v_k: RP_1 and RP_2. RP_1 is composed of vertices v_1 to v_{k-1}, and RP_2 is composed of vertices v_{k+1} to v_n. As the size of RP_1 is $k-1$ and the size of RP_2 is $n-k$, we obtain that $RP_1 = P_{k-1}$ et $RP_2 = P_{n-k}$.

It follows from these remarks that a recursive expression of $\mathbb{E}(MDG(P_n))$ is:

$$\mathbb{E}(MDG(P_0)) = \mathbb{E}(MDG(P_1)) = 0 \text{ and } \mathbb{E}(MDG(P_2)) = 1$$

$$\mathbb{E}(MDG(P_n)) = \sum_{k=2}^{n-1} \frac{1}{n-2} \cdot (1 + \mathbb{E}\left(MDG\left(P_{k-1}\right) + \mathbb{E}\left(MDG\left(P_{n-k}\right)\right)\right))$$

As $\sum_{k=2}^{n-1} \mathbb{E}(MDG(P_{k-1})) = \sum_{k=2}^{n-1} \mathbb{E}(MDG(P_{n-k}))$ and as $\mathbb{E}(MDG(P_1)) = 0$, we can reformulate our expression as follows:

$$\mathbb{E}(MDG(P_n)) = \sum_{k=2}^{n-1} \frac{1}{n-2} + 2 \cdot \sum_{k=2}^{n-2} \frac{1}{n-2} \cdot \mathbb{E}(MDG(P_k))$$

$$= 1 + \frac{2}{n-2} \sum_{k=2}^{n-2} \mathbb{E}(MDG(P_k))$$

□

LEMMA 7. *For any $n \geq 3$:*

$$\mathbb{E}(MDG(P_n)) = 1 + \frac{2}{n-2} \sum_{k=2}^{n-2} \mathbb{E}(MDG(P_k)) = \frac{n-1}{2} + \frac{n-1}{2} \sum_{p=0}^{n-1} \frac{(-2)^p}{p!} + \sum_{p=0}^{n-2} \frac{(-2)^p}{p!}$$

PROOF. We note $u_n = \mathbb{E}(MDG(P_n))$ for simplicity.

Let $u_n = 1 + \frac{2}{n-2} \sum_{k=2}^{n-2} u_k$, with $u_0 = u_1 = 0$, and $u_2 = 1$ (from Lemma 6).

First, we will show that $u_n = \frac{n-3}{n-2} u_{n-1} + \frac{2}{n-2} u_{n-2} + \frac{1}{n-2}$. It suffices to note that:

$$\frac{n-2}{n-3} u_n = \frac{n-2}{n-3} + \frac{2}{n-3} \sum_{k=2}^{n-2} u_k$$

$$= \frac{n-2}{n-3} + \frac{2}{n-3} \sum_{k=2}^{n-3} u_k + \frac{2}{n-3} u_{n-2}$$

$$= 1 + \frac{1}{n-3} + \frac{2}{n-3} \sum_{k=2}^{n-3} u_k + \frac{2}{n-3} u_{n-2}$$

$$= u_{n-1} + \frac{1}{n-3} + \frac{2}{n-3} u_{n-2}$$

hence

$$u_n = \frac{n-3}{n-2} u_{n-1} + \frac{2}{n-2} u_{n-2} + \frac{1}{n-2}$$

We will show now the result claimed in the lemma by induction. It is easy to show that the equality is true for $n = 3$. We assume that the equality is true up to rank $n-1$ and we verify it at rank n. We have:

$$u_n = \frac{n-3}{n-2} u_{n-1} + \frac{2}{n-2} u_{n-2} + \frac{1}{n-2}$$

Noting $b_n = \sum_{p=0}^{n} \frac{(-2)^p}{p!}$ and by induction hypothesis, we obtain:
$u_n = \frac{n-3}{n-2} \left(\frac{n}{2} - 1 + \left(\frac{n}{2} - 1 \right) b_{n-2} + b_{n-3} \right) + \frac{2}{n-2} \left(\frac{n}{2} - \frac{3}{2} + \left(\frac{n}{2} - \frac{3}{2} \right) b_{n-3} + b_{n-4} \right) + \frac{1}{n-2}$

$u_n = \frac{(n-3)n}{2(n-2)} - \frac{n-3}{n-2} + \left(\frac{(n-3)n}{2(n-2)} - \frac{n-3}{n-2}\right) b_{n-2} + \frac{n-3}{n-2} b_{n-3} + \frac{n-3}{n-2} + \frac{n-3}{n-2} b_{n-3} + \frac{2}{n-2} b_{n-4} + \frac{1}{n-2}$

$$u_n = \frac{n-3}{2} + \frac{n-3}{2} b_{n-2} + \frac{n-3}{n-2} b_{n-3} + \frac{n-3}{n-2} + \frac{n-3}{n-2} b_{n-3} + \frac{2}{n-2} b_{n-4} + \frac{1}{n-2}$$

$$u_n = \frac{n-1}{2} + \frac{n-3}{2} b_{n-2} + b_{n-3} \frac{n-3}{n-2} 2 + \frac{2}{n-2} b_{n-4}$$

To prove the result, it suffices to show that

$$Z = \frac{n-1}{2} + \frac{n-3}{2} b_{n-2} + b_{n-3} \frac{n-3}{n-2} 2 + \frac{2}{n-2} b_{n-4} - \left(\frac{n-1}{2} + \frac{n-1}{2} b_{n-1} + b_{n-2}\right) = 0$$

After simplifications, we obtain:

$$Z = \frac{n-5}{2} b_{n-2} + b_{n-3} \frac{n-3}{n-2} 2 + \frac{2}{n-2} b_{n-4} - \frac{n-1}{2} b_{n-1}$$

Noticing that $b_{n-k} = b_n - \sum_{j=n-k+1}^{n} a_j$, with $a_j = \frac{(-2)^p}{j!}$, we have:
$Z = \frac{n-5}{2} (b_{n-1} - a_{n-1}) + (b_{n-1} - a_{n-1} - a_{n-2}) \frac{n-3}{n-2} 2 + \frac{2}{n-2} (b_{n-1} - a_{n-1} - a_{n-2} - a_{n-3}) - \frac{n-1}{2} b_{n-1}$

$Z = b_{n-1} \left(\frac{n-5}{2} + \frac{n-3}{n-2} 2 + \frac{2}{n-2} - \frac{n-1}{2}\right) - a_{n-1} \left(\frac{n-5}{2} + \frac{n-3}{n-2} 2 + \frac{2}{n-2}\right) - a_{n-2} \left(\frac{n-3}{n-2} 2 + \frac{2}{n-2}\right) - \frac{2}{n-2} a_{n-3}$

$$Z = -\frac{n-1}{2} a_{n-1} - 2a_{n-2} - \frac{2}{n-2} a_{n-3}$$

Noticing that $a_{n-1} = -\frac{n}{2} a_n$, we obtain:

$$Z = -\frac{n-1}{2} a_{n-1} - 2\left(-\frac{n-1}{2} a_{n-1}\right) - \frac{2}{n-2}\left(\frac{(n-1)(n-2)}{4} a_{n-1}\right)$$

$$Z = a_{n-1}\left(-\frac{n-1}{2} + n - 1 - \frac{n-1}{2}\right) = 0$$

which proves the result. □

THEOREM 8. $\mathbb{E}(\text{EDGE DELETION}(P_n)) = n - n \sum_{p=0}^{n-1} \frac{(-2)^p}{p!} - 2 \sum_{p=0}^{n-2} \frac{(-2)^p}{p!}$.

PROOF. The result is proved in the same way as MDG. □

3.2 Limits of the expected approximation ratio

In this section, we study the asymptotic behavior of these algorithms on P_n by calculating the limit of their approximation ratio when the size of the path tends to infinity.

LEMMA 9. *Let $opt(P_n) = \lfloor \frac{n}{2} \rfloor$ be the size of an optimal vertex cover on P_n and $A(P_n)$ the size of the solution returned by an algorithm A applied on P_n. We have:*

$$\mathbb{E}\left(\frac{A(P_n)}{opt(P_n)}\right) = \frac{\mathbb{E}\left(A(P_n)\right)}{opt(P_n)}$$

PROOF. See proof in [Birmelé et al. 2009]. □

THEOREM 10. *The limit of the expected approximation ratio of* DEPTH FIRST SEARCH *on P_n when n tends to infinity is* 2.

PROOF. By lemma 9, and noticing that $\mathbb{E}(DFS(P_n)) = n - 2 + \frac{2}{n}$ and $opt(P_n) = \lfloor \frac{n}{2} \rfloor$, we obtain directly the result. □

THEOREM 11. *The limit of the expected approximation ratio of* LISTLEFT *on P_n when n tends to infinity is $\frac{4}{3}$.*

PROOF. The limit of the expected approximation ratio is obtained easily by using Theorem 3 and noticing that

$$\frac{2}{3}\frac{2n-1}{n-1} = \frac{\frac{2n-1}{3}}{\frac{n-1}{2}} \geq \frac{\frac{2n-1}{3}}{\lfloor \frac{n}{2} \rfloor} \geq \frac{\frac{2n-1}{3}}{\frac{n}{2}} = \frac{2}{3}\frac{2n-1}{n}.$$

□

The following result has been shown in [Birmelé et al. 2009].

THEOREM 12. *The limit of the expected approximation ratio of* LISTRIGHT *on P_n when n tends to infinity is $1 + e^{-2}$.*

PROOF. See proof in [Birmelé et al. 2009]. □

By applying similar techniques used in [Birmelé et al. 2009], we can show the limit of the expected approximation ration of MAXIMUM DEGREE GREEDY and EDGE DELETION. We only present the proof for MAXIMUM DEGREE GREEDY here.

THEOREM 13. *The limit of the expected approximation ratio of* MAXIMUM DEGREE GREEDY *on P_n when n tends to infinity is $1 + e^{-2}$.*

PROOF. It follows from Lemma 9 that:
$\lim_{n\to\infty} \mathbb{E}\left(\frac{MDG(P_n)}{OPT(P_n)}\right) = \lim_{n\to\infty} \frac{\mathbb{E}(MDG(P_n))}{OPT(P_n)} = \lim_{n\to\infty} \frac{\mathbb{E}(MDG(P_n))}{\lfloor \frac{n}{2} \rfloor}$.

Using Theorem 5 and with $b_n = \sum_{p=0}^{n} \frac{(-2)^p}{p!}$, we have

$$\mathbb{E}(MDG(P_n)) = \frac{n-1}{2} + \frac{n-1}{2} b_{n-1} + b_{n-2}$$

and

$$\frac{\frac{n-1}{2} + \frac{n-1}{2} b_{n-1} + b_{n-2}}{\frac{n}{2}} \leq \frac{\frac{n-1}{2} + \frac{n-1}{2} b_{n-1} + b_{n-2}}{\lfloor \frac{n}{2} \rfloor} \leq \frac{\frac{n-1}{2} + \frac{n-1}{2} b_{n-1} + b_{n-2}}{\frac{n-1}{2}}$$

Noticing that $e^{-2} = \sum_{k=0}^{\infty} \frac{(-2)^k}{k!}$ ([Abramowitz and Stegun 1964]), we calculate the limit of the left side:

$$\lim_{n\to\infty} \frac{\frac{n-1}{2} + \frac{n-1}{2} b_{n-1} + b_{n-2}}{\frac{n}{2}} = \lim_{n\to\infty} \frac{n - 1 + (n-1) b_{n-1} + 2 b_{n-2}}{n}$$

$$= 1 + \lim_{n\to\infty} \frac{n-1}{n} b_{n-1} + \lim_{n\to\infty} \frac{2}{n} b_{n-2} = 1 + e^{-2}$$

In the same way, we calculate the limit of the right side:

$$\lim_{n\to\infty} \frac{\frac{n-1}{2} + \frac{n-1}{2} b_{n-1} + b_{n-2}}{\frac{n-1}{2}} = \lim_{n\to\infty} \frac{n-1+(n-1)b_{n-1}+2b_{n-2}}{n-1}$$

$$= 1 + \lim_{n\to\infty} \frac{n-1}{n-1} b_{n-1} + \lim_{n\to\infty} \frac{2}{n-1} b_{n-2} = 1 + e^{-2}$$

which proves the result. □

THEOREM 14. *The limit of the expected approximation ratio of* EDGE DELETION *on P_n when n tends to infinity is $2 - 2e^{-2}$.*

PROOF. The result is proved in the same way as Theorem 13. □

Algorithm	Limit of the expectation of the approxiamtion ratio
GREEDY INDEPENDENT COVER	1
MAXIMUM DEGREE GREEDY	$1 + e^{-2} \approx 1.13$
LISTRIGHT	$1 + e^{-2} \approx 1.13$
LISTLEFT	$\frac{4}{3} \approx 1.33$
EDGE DELETION	$2 - 2e^{-2} \approx 1.73$
DEPTH FIRST SEARCH	2

Table I: Summary of the theoretical results obtained on paths

These different results show that the algorithms with the worst approximation ratios can achieve good results on graphs for which at least half of vertices are in the solution. At the opposite, the algorithms that have a constant approximation ratio, which are most frequently cited in the literature (eg EDGE DELETION) do not provide the best solutions.

4. EXPERIMENTAL ANALYSIS

In the previous section, we conducted an analytical comparative study of the behavior of different algorithms on paths. As it is difficult to get such analytical results on any graph we decided to conduct an experimental comparison of the average quality of these algorithms. In this section, we will describe the measure which seems to be the most appropriate to capture the average quality of algorithms. Subsequently, we will define the methodology we followed and at the end, we will present the results obtained on different samples of graphs.

What do we measure?

Let G be any graph with n vertices and let $OPT(G)$ be the size of an optimal vertex cover of G. We note $\mathbb{E}(A(G))$ the average size of vertex covers constructed when executing algorithm A several times on graph G. As we want to know if $\mathbb{E}(A(G))$ if far from $OPT(G)$, we calculate $\mathbb{E}(A(G)) - OPT(G)$. As graphs (of different size) are grouped in samples (having the same particular/structural properties), it is not

obvious to compare the algorithms just on this criteria since we should compare them graph by graph (some samples contain more than 4000 graphs). We need a way to "normalize" these values. To do that, we know that in G all the possible vertex covers have size between $OPT(G)$ and n. We define here the *percentage of error* of algorithm A on graph G, given by the following formula:

$$100.\frac{\mathbb{E}(A(G)) - OPT(G)}{n - OPT(G)}$$

(this measure comes from ideas similar to the ones of the differential approximation ratio (see [Monnot et al. 2003] for a detailed presentation)).

4.1 Experimental method

In this sub-section we present our experimental approach, our tools and the samples of graphs on which we conducted our tests.

Randomness and execution time. In the initial description of the algorithms (in Section 2.1) the mechanisms to make the choices are not specified. As our software was developed in Java, we used the Java random generator to make the equiprobable different "local" choices (otherwise, each execution of the same algorithm on the same graph would always return the same size, see discussions in Section 2.2). However, in a real application, a programmer would implement a deterministic and fast choice with optimal data structures. This is not what we did here since we need variability between executions. Thus, we do not give the computation times of various algorithms or details about the machine on which we ran our experiments. However, we know that all algorithms have polynomial time complexity (at most in $O(nm)$) with appropriated data structures, are easy to implement and do not use "hidden" tools like LP solver (this is why we study them, see Introduction of the paper).

"Sorted versions" of ListRight and ListLeft. In [Avis and Imamura 2007], the authors used the ListLeft algorithm with lists that are sorted by decreasing order of their degrees. In order to assess the influence of sorting lists on the performances of lists algorithms, we will consider two versions of each list algorithm. The sorted versions, called Sorted ListLeft and Sorted ListRight and unsorted versions called ListLeft and ListRight. The algorithms Sorted ListLeft and Sorted ListRight require the generation of sorted lists. To get them, we had the choice between two methods: selecting one list uniformly among all possible lists (i.e. $n!$, with n the size of the graph) and then sort the list or select a list uniformly among all sorted lists. We chose the first method for its simplicity and to consider Sorted ListLeft and Sorted ListRight in the same manner than ListLeft and ListRight because this allows us, given a list, to compare the four algorithms, which is quite interesting.

Experimental method. The graphs to be tested are grouped in samples (see after for a description of the samples). For each graph G of each sample, we used LPSOLVE or Cplex to calculate $OPT(G)$, the size of an optimal vertex cover. For each sample, we made $10,000$ iterations of each algorithm on each instance of the

sample. Then we calculated the average of these $10,000$ iterations for each algorithm. These values allows us to calculate the percentage of error of each algorithm for each graph G.

The samples. Since computing the optimal solution can be very long (even using fast solvers like Cplex), we are restricted to graphs with less than 1000 vertices.

Erdös-Renyi Random Graphs. We generated 4500 random graphs using the Erdös-Renyi model. More precisely, for each size of graph n, $n \in \{20, 40, 60, 80, 100\}$ and for each probability p with $p \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, we generated 100 graphs $G_{n,p}$.

Remark about the number of iterations. In [Delbot 2009], the first author provided two formulas to compute the exact expectation of the size of solutions returned by algorithms ListLeft and ListRight on Erdös-Renyi graphs. We compared these two formulas to the experimental results obtained by these two algorithms on this sample and we showed that they match very well. It indicates that our sample of random graphs is sufficiently representative of Erdös-Reyni graphs. Moreover, the fact that our algorithms behave as expected indicates that the number of iterations that we have made is large enough. Hence, we assume that this will be the same for the other algorithms.

Benchmarks with Hidden Optimum Solutions (BHOSLIB). These benchmarks can be found at "http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm" and are composed of hard instances generated with the Model *RB* (see [Xu and Li 2006]). Instances are generated as follow:

(1) Generate n disjoint cliques, each of which has n^α vertices (where $\alpha > 0$ is a constant);
(2) Randomly select two different cliques and then generate without repetitions $pn^{2\alpha}$ random edges between these two cliques (where $0 < p < 1$ is a constant);
(3) Run Step 2 (with repetitions) for another $rn \ln n - 1$ times (where $r > 0$ is a constant).

In [Xu et al. 2007], the authors show how to choose appropriate values of p, r et α. Benchmarks obtained from this method are the following[2]:

—5 instances with 450 vertices (30 cliques), with OPT = 420;
—5 instances with 595 vertices (35 cliques), with OPT = 560;
—5 instances with 760 vertices (40 cliques), with OPT = 720;
—5 instances with 945 vertices (45 cliques), with OPT = 900;
—5 instances with 1150 vertices (50 cliques), with OPT = 1100;
—5 instances with 1272 vertices (53 cliques), with OPT = 1219;
—5 instances with 1400 vertices (56 cliques), with OPT = 1344;

[2]For each instance, the size of the minimum vertex cover is known, but the solution itself is not given.

—5 instances with 1534 vertices (59 cliques), with OPT = 1475;

Trees. We compared the algorithms on trees. These graphs have a low density, are connected and it is easy to compute an optimal solution (for example, with Greedy Independent Cover). Instances are generated as follow:

(1) Start with a complete graph with $n > 3$ vertices;
(2) Delete a random edge such that the resulting graph is connected;
(3) Run step 2 while there is more than $n - 1$ edges.

With this method, and for $n = 50$, 100, 500 and 1000, we generated 1000 trees (for a total of 4000 trees).

Average worst case. Graphs that trap an algorithm. Given an algorithm A and a measure (approximation ratio for example), we say that a graph G *traps* A if A, executed on G, leads to the worst possible measure (or at least to a "bad" measure). When it was possible, we generated instances that trap an algorithm on average and in terms of the percentage of error. The interest of this sample is twofold. On the one hand, it is necessary to show that different algorithms may get bad results in average on at least one class of graphs, and it is interesting, secondly, to observe the behavior of the other algorithms on these particular instances.

—For Sorted ListLeft, we used the graphs that we call $ProW$ presented in [Delbot and Laforest 2008] (see Fig. 1 for $N = 3$ for example). On these graphs, this algorithm always returns a solution of size exactly $\frac{N+1}{2}|OPT|$, with $N \geq 2$ the dimension of the graph.
—For Sorted ListRight, we used a modification of graphs $ProW$ in order to obtain the same behavior. We link each vertex of F_1 to each vertex of V_2 (see Fig. 2 for $N = 3$ for example).

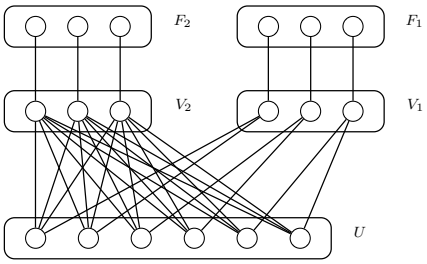

Fig. 1: Graph $ProW$ with $N = 3$.

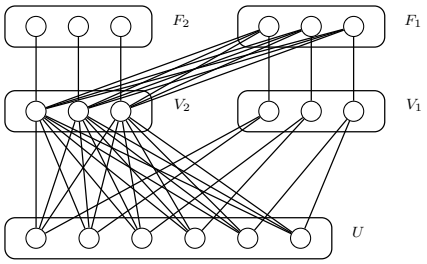

Fig. 2: Modified graph $ProW$ with $N = 3$.

—For Edge Deletion, we used complete bipartite graphs $K_{a,a}$ with the two sets of vertices of equal size a, because each execution of this algorithm returns a perfect matching, which is the worst solution.
—For Maximum Degree Greedy, we used the $Anti - MDG$ graphs presented in [Papadimitriou and Yannakakis 1988] for example without the vertices of degree

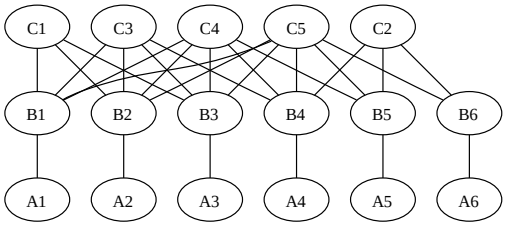

Fig. 3: Example of a modified $Anti - MDG$ graph of dimension 6.

2, in order to force any execution to return the worst case (see Fig. 3 for an example).

—For Greedy Independent Cover, we used the graphs presented in [Avis and Imamura 2007]. On these graphs, it always returns a $\frac{\sqrt{\Delta}}{2}$ approximated solution.

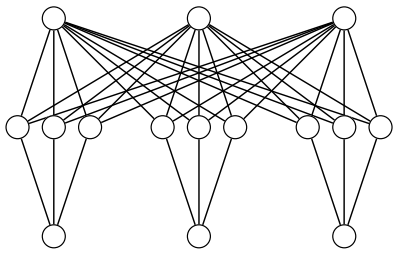

Fig. 4: Example of trap graph for Greedy Independent Cover with $\Delta = 9$.

—For ListLeft, we did not need to use a specific class of graphs because this algorithm performs bad on most instances previously presented.

—Finding a class of graphs that trap an algorithm in average is not a simple task. We shown in [Birmelé et al. 2009] that the average size returned by algorithm ListRight on any graph G is at most $2OPT(G)$; this average approximation ratio tends to be reached for stars when the number of leaves tends to infinity. However, the percentage of error of ListRight on the stars is very close to 0 as its average performance is very far from the worst case (all the leaves and the center of the star). We did not find graphs trapping ListRight for the percentage of error parameter. For this reason, we will evaluate this algorithm only on the instances that trap the other algorithms.

Finally, our average worst case sample contains:

—5 modified $Anti - MDG$ graphs of dimension from 30 to 34;

—5 graphs of Avis and Imamura of dimension from 30 à 34;

—5 $ProW$ graphs with N from 30 à 34;

—5 modified $ProwW$ graphs with N from 30 à 34;

—5 complete bipartite graphs: $K_{10,10}$, $K_{15,15}$, $K_{20,20}$, $K_{25,25}$, $K_{30,30}$.

Regular and near-regular graphs. We also wanted to compare the algorithms on graphs with a regular structure such as grids, tori, hypercubes and random regular graphs. These graphs have an optimal solution that contains half of the vertices (± one vertex for grids and tori). This sample contains:

—all grids and tori of size 2×2 to 20×20;

—hypercubes of dimension 3 to 12;

—36 random regular graphs with a number of vertices between 28 and 98. The degree of these graphs varies from 3 to 8.

4.2 Experimental results

The results are grouped by sample, with one sample per page. Each page contains 8 figures representing the performance of each algorithm. The horizontal axis of each figure represents the rounded up percentage of error obtained by the algorithm to an instance of the sample and the vertical axis represents the number of instances of the sample that had received this value. See Figure 5 for an example and explanation.

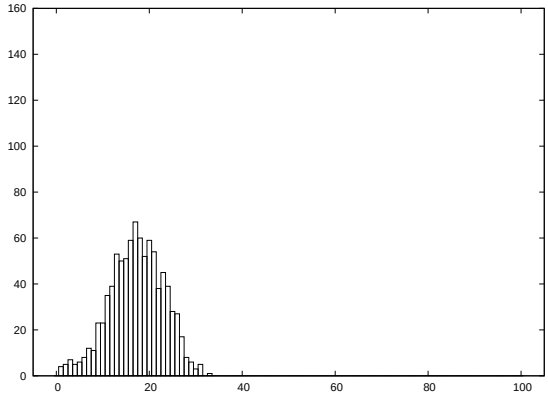

Fig. 5: Example of image obtained for the MDG algorithm when applied 10,000 times to each graph of the Erdös-Renyi sample. The best percentage of error reach 0 while the worst is less than 38. This algorithm obtain a percentage of error between 10% and 25% for the majority of the instances of the sample.

—Figures 6, 7, 8, 9, 10, 11, 12 and 13 are obtained by applying the algorithms on the random Erdös-Renyi graphs sample.

—Figures 14, 15, 16, 17, 18, 19, 20 and 21 are obtained by applying the algorithms on the trees sample.

—Figures 22, 23, 24, 25, 26, 27, 28 and 29 are obtained by applying the algorithms on the benchmarks BHOSLIB.

—Figures 30, 31, 32, 33, 34, 35, 36 and 37 are obtained by applying the algorithms on the average worst case graphs sample.

—Figures 38, 39, 40, 41, 42, 43, 44 and 45 are obtained by applying the algorithms on the regular graphs sample.

To understand the overall performance of an algorithm on a sample, it is interesting to know the extreme values of the percentage of error. Table II contains, for each sample and each algorithm the best and the worst percentage of error obtained on the sample.

Sample	GIC	MDG	SLL	SLR	LL	LR	ED	DFS
Erdös-Renyi	0-25	1-33	25-88	0-43	63-82	13-42	68-97	36-91
Trees	0-0	0-9	2-19	0-6	29-42	7-17	46-67	18-48
BHOSLIB	15-20	30-44	74-90	24-36	81-85	32-34	85-88	89-98
Regular graphs	0-8	0-32	0-83	0-55	16-83	0-55	68-100	50-99
Average worst case graphs	0-94	0-70	12-98	0-95	34-97	0-18	6-100	6-100

Table II: Summary of the performances obtained by each algorithm for each sample. The first number represents the best percentage of error obtained on the sample and the second represents the worst.

The average size of solutions is an important property to capture when studying the average performance of an algorithm. Table III contains, for each sample and each algorithm, the average size of solutions.

Sample	LL	LR	SLL	SLR	ED	DFS	MDG	GIC
Erdös-Renyi ($n = 100$)	96.85	91.34	94.64	90.07	97.59	96.78	89.79	89.0
Trees ($n = 100$)	62.56	49.01	47.54	43.09	73.98	59.22	43.73	42.35
BHOSLIB	1005.59	982.39	1006.07	980.32	1006.95	1010.52	983.59	975.4
Regular graphs	87.64	69.89	88.89	69.44	100.06	107.56	66.1	57.86
Average worst case graphs	610.48	133.15	598.14	413.82	153.35	163.95	99.7	242.66

Table III: Summary of the average size of solutions obtained by each algorithm for each sample.

Table IV contains, for each sample and each algorithm, the smallest and et the largest variance obtained on the sample.

	Erdös-Renyi		BHOSLIB		Trees		Reg. graphs		Av. worst case graphs	
	V_{min}	V_{max}	V_{min}	V_{max}	V_{min}	V_{max}	V_{min}	V_{max}	V_{min}	V_{max}
LL	0.8	5.39	2.28	4.15	7.01	14.95	0.0	71.97	1.36	1319.45
LR	0.0	7.69	1.75	3.29	2.21	11.37	0.0	318.06	0.0	33242.0
SLL	0.0	2.21	0.0	0.5	0.12	3.05	0.0	72.55	0.0	31.16
SLR	0.0	6.16	0.0	3.38	0.0	1.55	0.0	318.77	0.0	17.61
ED	0.29	6.31	2.80	5.13	4.97	13.33	0.0	74.87	0.0	51.23
DFS	0.0	3.15	0.0	0.01	0.19	0.25	0.0	5.24	0.0	0.19
MDG	0.0	2.08	0.7	2.32	0.0	1.81	0.0	9246.3	0.0	12.76
GIC	0.0	3.59	0.07	1.97	0.0	0.0	0.0	11.91	0.0	0.0

Table IV: Summary of the variances obtained by each algorithm for each sample. The first number represents the smallest variance obtained for the sample and the second represents the largest.

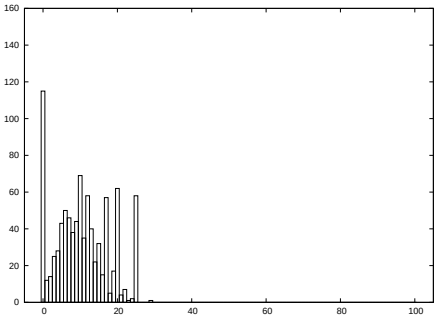

Fig. 6: Greedy Independent Cover

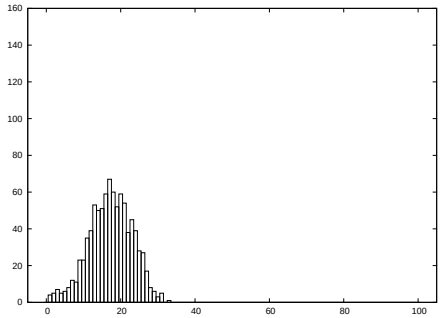

Fig. 7: Maximum Degree Greedy

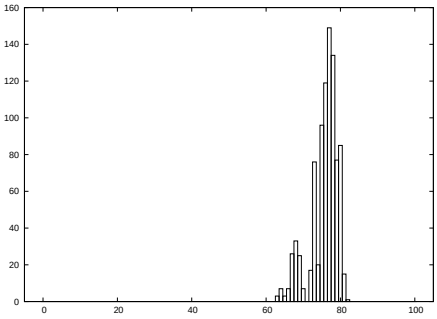

Fig. 8: ListLeft

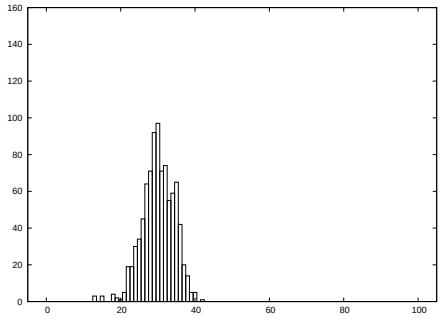

Fig. 9: ListRight

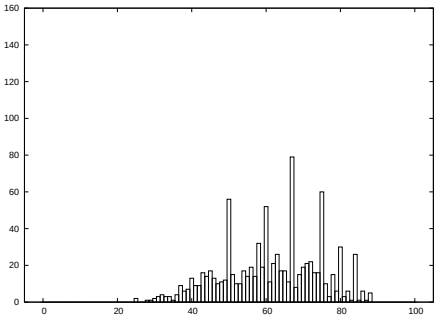

Fig. 10: Sorted ListLeft

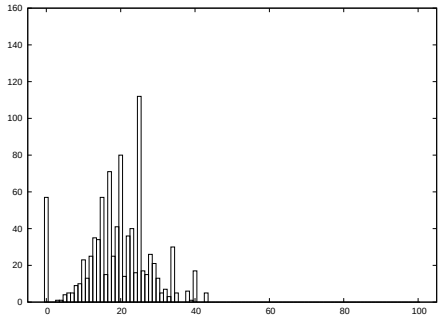

Fig. 11: Sorted ListRight

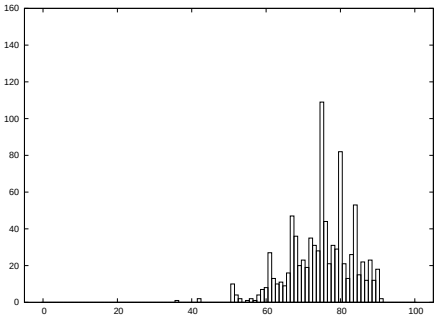

Fig. 12: Depth First Search

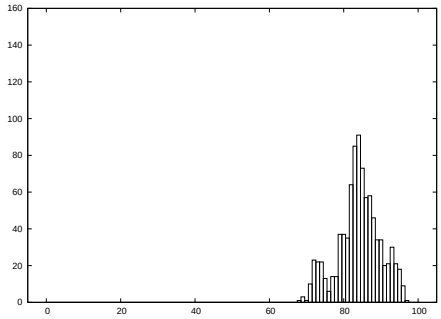

Fig. 13: Edge Deletion

Erdös-Renyi random graphs sample

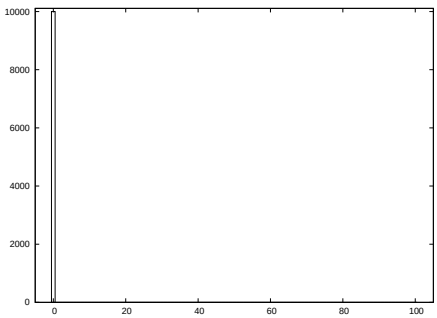

Fig. 14: Greedy Independent Cover

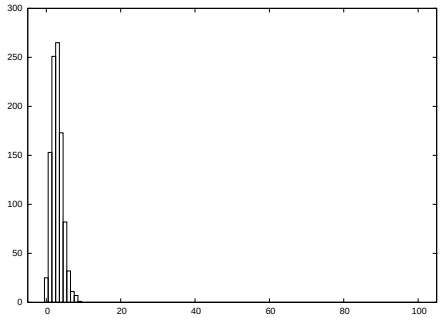

Fig. 15: Maximum Degree Greedy

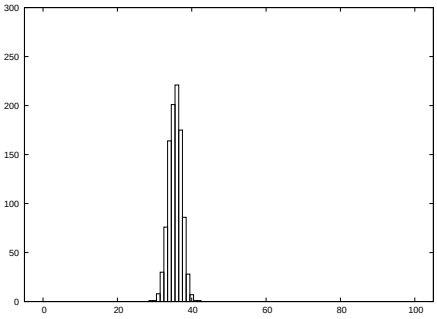

Fig. 16: ListLeft

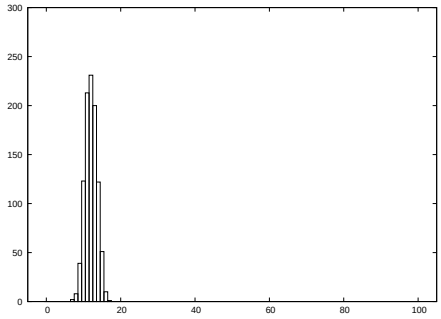

Fig. 17: ListRight

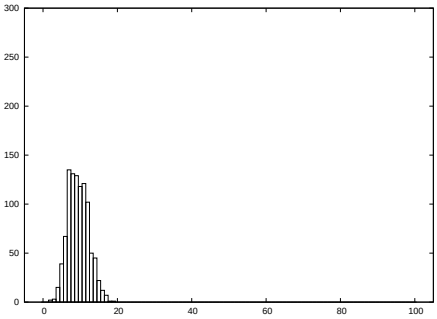

Fig. 18: Sorted ListLeft

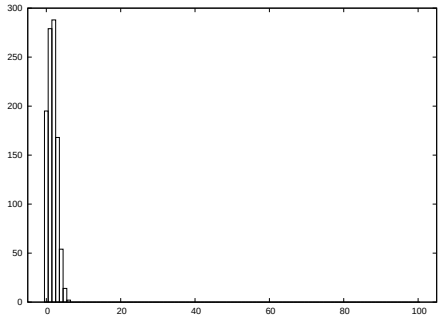

Fig. 19: Sorted ListRight

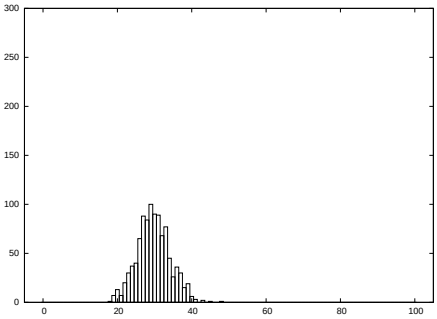

Fig. 20: Depth First Search

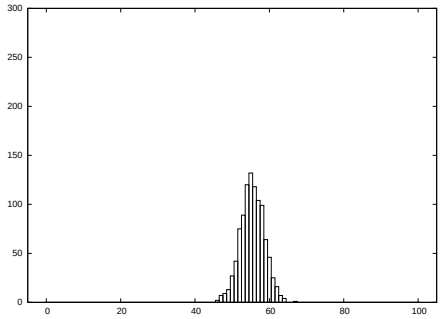

Fig. 21: Edge Deletion

Trees Sample

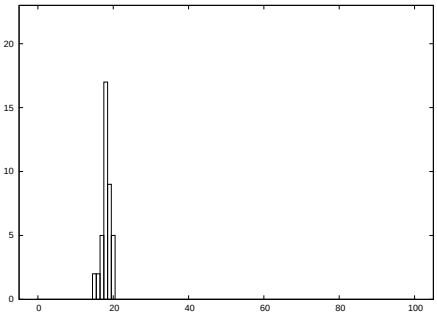

Fig. 22: Greedy Independent Cover

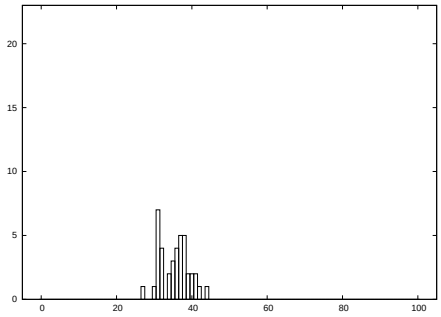

Fig. 23: Maximum Degree Greedy

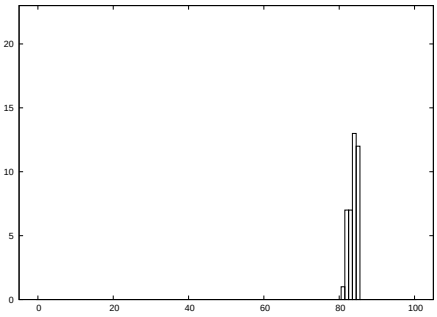

Fig. 24: ListLeft

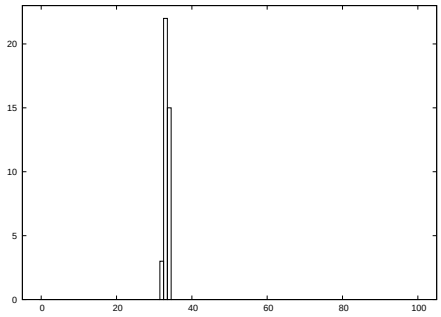

Fig. 25: ListRight

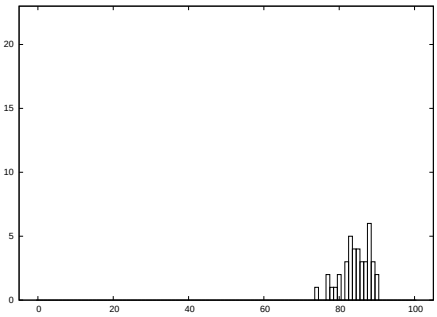

Fig. 26: Sorted ListLeft

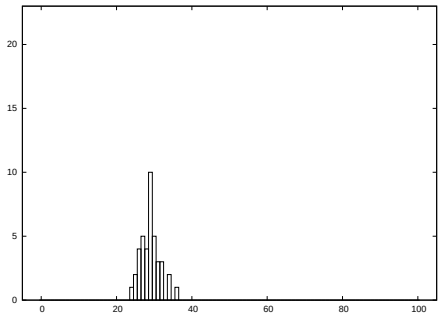

Fig. 27: Sorted ListRight

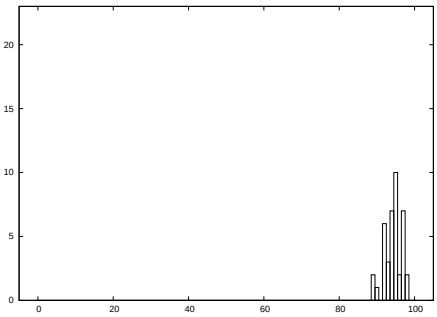

Fig. 28: Depth First Search

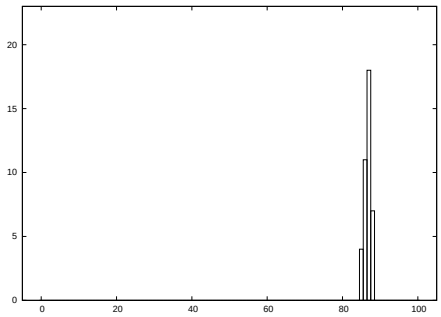

Fig. 29: Edge Deletion

BHOSLIB sample

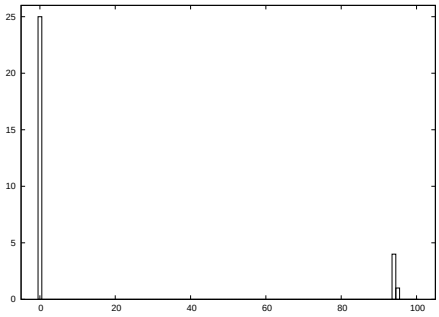

Fig. 30: Greedy Independent Cover

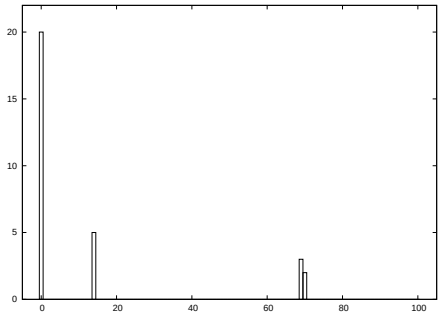

Fig. 31: Maximum Degree Greedy

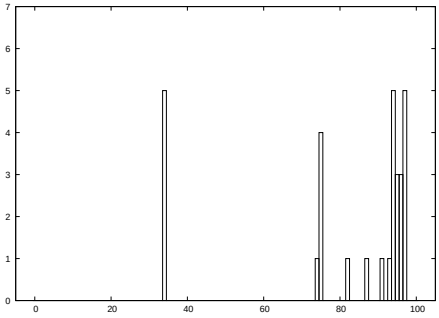

Fig. 32: ListLeft

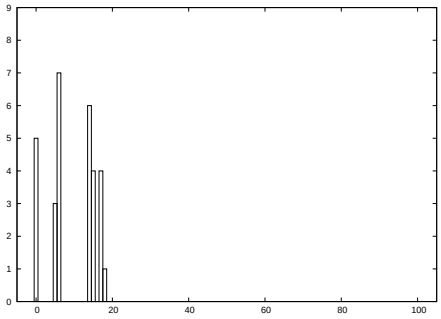

Fig. 33: ListRight

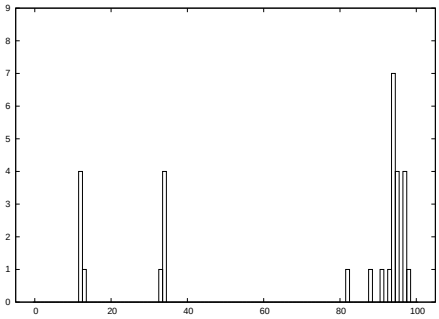

Fig. 34: Sorted ListLeft

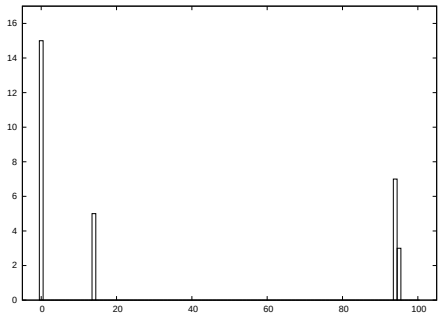

Fig. 35: Sorted ListRight

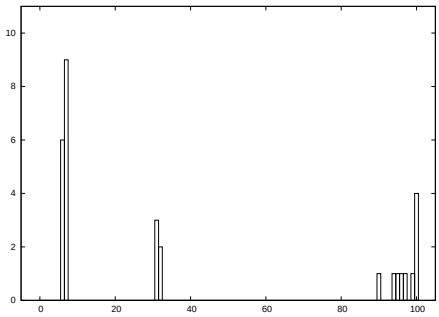

Fig. 36: Depth First Search

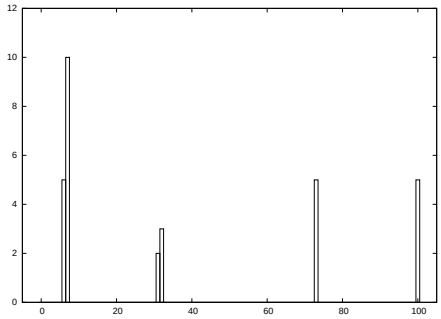

Fig. 37: Edge Deletion

Average worst case sample

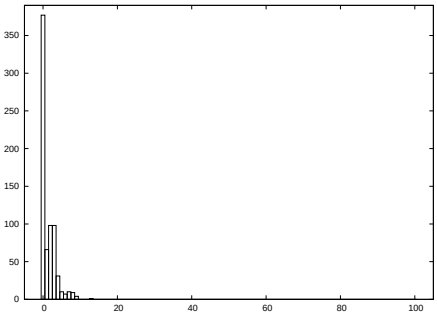

Fig. 38: Greedy Independent Cover

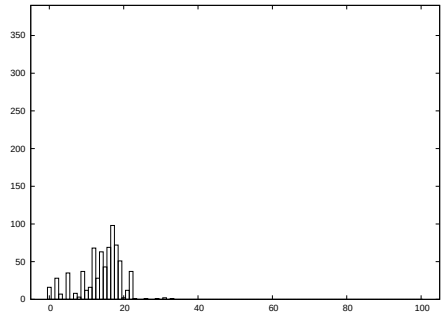

Fig. 39: Maximum Degree Greedy

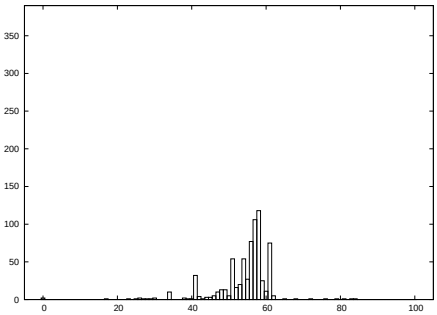

Fig. 40: ListLeft

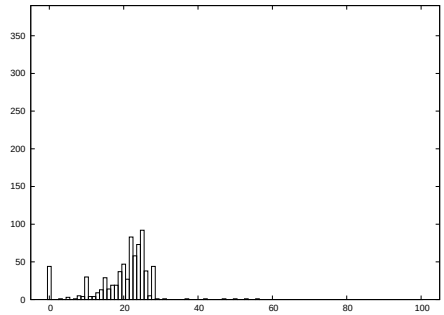

Fig. 41: ListRight

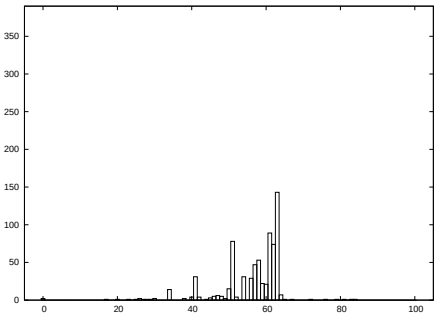

Fig. 42: Sorted ListLeft

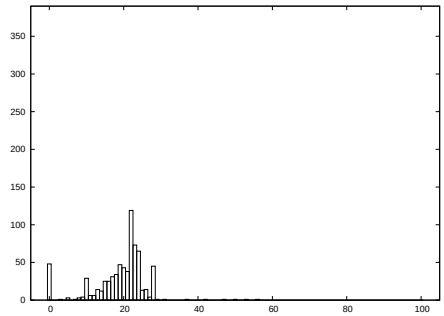

Fig. 43: Sorted ListRight

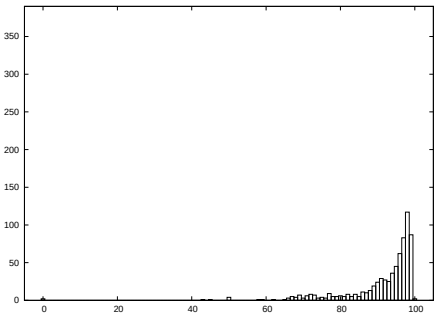

Fig. 44: Depth First Search

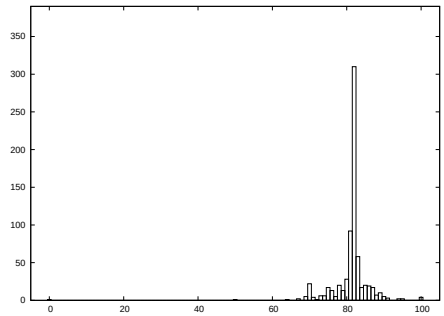

Fig. 45: Edge Deletion

Regular graphs sample

4.3 Analysis of the results and comments

4.3.1 Greedy Independent Cover *and* Maximum Degree Greedy. The most visible result is the very good performance of the algorithm Greedy Independent Cover. The percentage of error of this algorithm is better than all the other algorithms, for all instances of our samples except for the 5 $Anti - GIC$ graphs. This algorithm is optimal on average for many instances, like the Erdös-Renyi graphs with a low probability of edge, trees and a large part of regular graphs. Its percentage of error never exceeds 25% (except for the 5 $Anti-GIC$ graphs) and its variance is always small, showing that it is an excellent algorithm. Even if it is worse than Greedy Independent Cover, Maximum Degree Greedy is also very good. Its percentage of error never exceeds 44% (except for the $Anti - MDG$ graphs) and it has, for each sample, one of the three best average solution sizes. Moreover, its variance is low (apart from the hypercubes of the regular graphs sample).

4.3.2 *List algorithms.* Even if the descriptions of ListLeft and ListRight are similar, their performances are very different. By observing the results in Table II, we can remark that for 4 samples, the worst percentage of error obtained by ListRight is better than the best percentage of error obtained by ListLeft. The percentage of error of ListRight never exceeds 55% and the variances of the two algorithms are quite low, except for certain specific instances such as $Anti - GIC$ or hypercubes. Tables III and IV indicates that ListRight is a good algorithm (even if it is not the best). ListLeft is rather bad since most percentages of error are above 50%. Sorting the lists by decreasing degrees seems to be a good idea since this yields to better results (compared to the non-sorted versions). However, sorting the vertices has the effect to decrease the quality of returned solutions for some instances of the random graphs sample and the BHOSLIB sample.

4.3.3 Edge Deletion *and* Depth First Search. These two algorithms have similar performances, even if Edge Deletion is slightly worse. Edge Deletion performs worse than all other algorithms on the Erdös-Renyi graphs samples and BHOSLIB benchmarks, and all algorithms except Depth First Search for regular graphs and the 4 algorithms Greedy Independent Cover, Maximum Degree Greedy, Sorted ListRight and ListRight on trees. Its percentage of error is never less than 46%, which shows that this algorithm returns only solutions of poor quality. Tables III and IV indicates that they are the two worst algorithms studied in this article.

4.3.4 *Synthesis of our study.* From these results and comments, we ranked the algorithms, sample by sample by using the following (informal) criteria:

"algorithm A is better than algorithm B on a given sample if most of the percentages of error obtained by algorithm A are less than those obtained by algorithm B."

From table V, we can conclude that the ranking of the 4 algorithms Greedy Independent Cover, Maximum Degree Greedy, Depth First Search and

Sample	1	2	3	4	5	6	7	8
Erdös-Renyi	GIC	MDG	SLR	LR	SLL	LL	DFS	ED
Trees	GIC	SLR	MDG	SLL	LR	DFS	LL	ED
BHOSLIB	GIC	SLR	LR	MDG	LL	SLL	ED	DFS
Regular graphs	GIC	MDG	SLR	LR	SLL	LL	DFS	ED
Average worst case graphs	GIC	MDG	SLR	LR	SLL	LL	DFS	ED

Table V: Table of rank of the algorithms by sample

Edge Deletion, based on experimental results is exactly **the opposite** of the one obtained using only the worst-case approximation ratio. Moreover, ListRight behaves very well, despite its large worst case approximation ratio.

5. CONCLUSION

In a first part of this paper, we evaluated analytically the expectation of 6 algorithms on paths: 2 algorithms with worst case approximation ratio 2 (Edge Deletion and Depth First Search), 2 list algorithms (ListLeft and ListRight) and 2 greedy algorithms with non-constant and large worst case approximation ratios (Maximum Degree Greedy and Greedy Independent Cover). As in paths no vertex cover has size more than 2 times an optimal one no algorithm is, *a priori*, penalized by a potential "pathological behavior" that would lead to solutions arbitrarily larger than an optimal one. This family of graphs is then interesting to make comparisons. We have proved that the algorithms having the best worst case approximation ratios are those who get the worst average results in paths. At the opposite we see that guarantying a worst case approximation ratios of 2 (like Edge Deletion and Depth First Search) has a "cost" in terms of structure of the returned solutions: they are in general non minimal for inclusion and this seems to have a negative impact on the expectation. To finish, it is known that Greedy Independent Cover is optimal on trees, and thus on paths.

In the second part of this paper, we conducted an experimental comparison of the quality of solutions returned by these 6 algorithms (plus 2 variants: Sorted ListLeft and Sorted ListRight). We proposed a measure (derived from the differential approximation ratio) to assess the average quality of solutions. Then, we made samples of different instances with different properties. It appears from this study that the two algorithms with approximation ratio of 2 return solutions of poor quality. Instead, the two greedy algorithms Greedy Independent Cover and Maximum Degree Greedy are very efficient and always return solutions of excellent quality, exept for graphs specifically designed to trap them.

The classification of Greedy Independent Cover, Maximum Degree Greedy, Depth First Search and Edge Deletion based on experimental results and those obtained on paths is exactly **the opposite** of the one obtained using only the worst-case approximation ratio.

Another spectacular result is that, despite a very bad worst case, the expectation of the size of solutions returned by ListRight (which is an online algorithm) seems

to be always less than or equal to the expectation of the size of solutions returned by Edge Deletion (which is offline and has an approximation ratio of 2).

From this study we can conclude that Greedy Independent Cover is the algorithm that obtains the best average performances. However, we exhibited special graphs for which it returns large solutions compared to optimal. It is therefore necessary to make a choice. Either we are seeking for the best solutions by agreeing to obtain solutions of poor quality in very rare cases (by choosing Greedy Independent Cover), or we accept solutions whose quality is slightly worse than Greedy Independent Cover but with a guarantee on average performance (by choosing ListRight), or we want the best guaranteed in worst case (by choosing Edge Deletion) while being aware that this choice comes implies a bad average quality of the returned solutions.

Perspectives. From a practical point of view the average behavior of the algorithms gives interesting information compared to the worst case approximation. However, we do not affirm that this average behavior is the *only* pertinent measure. We only say that it is a useful complement to the worst case analysis. It is pertinent only if it is significant. To measure the pertinence a more detailed analysis (based on variance for example) is required to complete the study. This is what we plan to do in the future.

Another perspective is to extend our study to other discrete optimization problems. For example, some *domination* problems in graphs are similar to vertex cover and could be investigated. More generally, we think that the approximation algorithms should also be evaluated and compared in average. Analytical and experimental methodology must be developed for that. We obtained preliminary results in this study and in [Delbot and Laforest 2008] and we plan to go further.

In this perspective, we are convinced that for any graph G, $\mathbb{E}(\text{ListRight}(G)) \leq \mathbb{E}(\text{Edge Deletion}(G))$. It would be interesting to give an analytical proof of that inequality or to show that graphs violating it are rare.

Acknowledgments. The authors thank the anonymous referees for they constructive comments.

REFERENCES

Abramowitz, M. and Stegun, I. A. 1964. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, ninth Dover printing, tenth GPO printing ed. Dover, New York.

Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., and Kann, V. 1999. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Avis, D. and Imamura, T. 2007. A list heuristic for vertex cover. *Operations Research Letters 35,* 2, 201–204.

Birmelé, E., Delbot, F., and Laforest, C. 2009. Mean analysis of an online algorithm for the vertex cover problem. *Information Processing Letters 109,* 9, 436–439.

Cardinal, J., Labbé, M., Langerman, S., Levy, E., and Mélot, H. 2005. A tight analysis of the maximal matching heuristic. In *Computing and Combinatorics: 11th Annual International Conference, COCOON 2005.* Lecture Notes in Computer Science, vol. 3595. Springer–Verlag, Kunming, China, 701 – 709.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. 2009. *Introduction to Algorithms*, 3rd ed. MIT Press.

Delbot, F. 2009. Au delà de l'évaluation en pire cas : comparaison et évaluation en moyenne de processus d'optimisation pour le problème du vertex cover et des arbres de connexion de groupes dynamiques. Ph.D. thesis, Université d'Évry Val d'Essonne.

Delbot, F. and Laforest, C. 2008. A better list heuristic for vertex cover. *Information Processing Letters 107,* 3-4, 125–127.

Demange, M. and Paschos, V. T. 2005. On-line vertex-covering. *Theoretical Computer Science 332,* 1-3, 83–108.

Garey, M. and Johnson, D. 1979. *Computers and Intractability.* Freeman and Co., New York.

Halldórsson, M. and Radhakrishnan, J. 1994. Greed is good: approximating independent sets in sparse and bounded-degree graphs. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing.* ACM, New York, NY, USA, 439–448.

Hochbaum, D. S., Ed. 1997. *Approximation algorithms for NP-hard problems.* PWS Publishing Compagny, Boston, MA, USA.

Khot, S. and Regev, O. 2008. Vertex cover might be hard to approximate to within $2-\epsilon$. *Journal of Computer and System Sciences 74,* 3, 335–349.

Monnot, J., Paschos, V. T., and Toulouse, S. 2003. *Polynomial approximation of hard NP-problems: local extremals and differential relation. (Approximation polynomiale des problèmes NP-difficiles: optima locaux et rapport différentiel.).* Hermes Science Publications.

Papadimitriou, C. and Yannakakis, M. 1988. Optimization, approximation, and complexity classes. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing.* ACM Press, New York, NY, USA, 229–234.

Roth-Korostensky, C. 2000. Algorithms for building multiple sequence alignments and evolutionary trees. Ph.D. thesis, ETH Zürich, Institute of Scientific Computing.

Savage, C. 1982. Depth-first search and the vertex cover problem. *Inf. Process. Lett. 14,* 5 (July), 233–235.

Stege, U. 2000. Resolving conflicts from problems in computational biology. Ph.D. thesis, ETH Zürich, Institute of Scientific Computing.

Vazirani, V. 2002. *Approximation algorithms.* 1st edition, Corr. 2nd printing. Springer.

Xu, K., Boussemart, F., Hemery, F., and Lecoutre, C. 2007. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artif. Intell. 171,* 8-9, 514–534.

Xu, K. and Li, W. 2006. Many hard examples in exact phase transitions. *Theor. Comput. Sci. 355,* 3, 291–302.

Zhang, Y., Ge, Q., Fleischer, R., Jiang, T., and Zhu, H. 2006. Approximating the minimum weight weak vertex cover. *Theoretical Computer Science 363,* 1, 99–105.