A greedy approach based algorithm for the vertex cover problem

Sushil Chandra Dimri, Kamlesh Chandra Purohit, Durgesh Pant

Abstract— The vertex cover problem is NP complete problem; we use approximation algorithms to find near optimal solution of the vertex cover problem. In this paper we are presenting a greedy algorithm for finding a near optimal cover for a given graph G = (V, E).The development of the algorithm is based on greedy approach and the graph is represented in form of its adjacency matrix. The proposed algorithm finds a minimum vertex cover in all known examples of graphs.

Index Terms— Minimum 7 keywords are mandatory, Keywords should closely reflect the topic and should optimally characterize the paper. Use about four key words or phrases in alphabetical order, separated by commas.

1 INTRODUCTION

There is a class of problems, whose exponential complexities have been established theoretically are known as NP problems. Designing polynomial time algorithms for such a class of problems is still open.[1 and 2] Due to the demand for solving such problems, Researchers are constantly attempting to provide solutions one after the other focusing the optimality by introducing several operators with salient features such as:

1. Reducing the computational complexity,
2. Randomizations etc.,

The vertex cover (VC) problem belongs to the class of NP-complete graph theoretical problems, which plays a central role in theoretical computer science and it has a numerous real life applications [3]. We are unlikely to find a polynomial-time algorithm for solving vertex-cover problem exactly. Vertex-cover exhibits a coverable–uncover able phase transition.

Branch-and-bound problem solver (BB), approximation algorithm, greedy algorithm simple genetic algorithm (GA), primal-dual based algorithm (PDB) and the Alom's algorithm. These approaches can be used to solve Vertex Cover problem. The results indicate that all algorithms give near optimal solutions. The performance differences of all algorithms on a graph are relatively small to obtain a vertex-cover.

Vertex cover problem is a NP-complete problem. If a problem is NP-complete, we are unlikely to find polynomial-time algorithm for solving it exactly, but this does not imply that all hope is lost. There are two approaches to getting around NP-completeness. First if the actual inputs are small, an algorithm with exponential running time may be perfectly satisfactory. Second, it may still possible to find near optimal solutions in polynomial time. In practice near optimality is often good enough. An algorithm that returns near-optimal solutions is called an approximation. [4, 5 and 6].

In computer science, the Vertex Cover Problem or Node Cover Problem is one of Karp's 21 NP-complete problems. It is often used in complexity theory to prove NP-hardness of more complicated problems. The classical minimum vertex-cover problem involves graph theory and finite combinatory and is categorized under the class of NP-complete problems in terms of its computational complexity [7]. Minimum vertex cover has attracted researchers and practitioners because of the NP-completeness and because many difficult real-life problems can be formulated as instances of the minimum vertex cover. Examples of the areas where the minimum vertex-cover problem occurs in real world applications are communications, civil and electrical engineering, and bioinformatics.
Definition: Consider a graph G = (V, E) where V and E are accordingly set of vertices and edges. A vertex cover of an undirected graph G is a subset V (c) of V such that if (u, v) is an edge of G, Then either u $\in$ V(c) or v $\in$ V(c) or both.

The size of a vertex cover is the number of vertices in it. The vertex cover problem is to find a vertex cover of minimum size in a given undirected graph. Such a vertex cover is called an optimal vertex cover. 'Coreman' describes an approximation algorithm with O (E) time for vertex cover problem. This algorithm finds the approximate solution.

There are two versions of the minimum vertex cover problem: the decision version and the optimization one. In the decision version, the task is to verify for a given graph whether there exists a vertex cover of a specified size. On the other hand, in the optimization version of this problem, the task is to find a vertex cover of minimum size. To illustrate minimum vertex cover, consider the problem of placing guards Hartmann, [8] in a museum where corridors in the museum corresponds one guard at the end of each corridor.

- *Sushil Chandra Dimri, Professor and Head of Computer Applications department in Graphic Era University. Esbb E-mail: sushil.dimri82@gmail.com*
- *Kamlesh Chandra Purohit, Assistant Professor in Computer Applications department in Graphic Era University. E-mail: kamleshpurohit80@gmail.com*
- *Durgesh Pant, Professor in Computer Science department Uttarakhand open university E-mail: pantdurgesh@yahoo.com*

The minimum vertex cover problem is also closely related to many other hard graph problems and so it interests the researchers in the field of design of optimization and approximation algorithms. For instance, the independent set problem, is similar to the minimum vertex cover problem because a minimum vertex cover defines a maximum independent set and vice versa. Another interesting problem that is closely related to the minimum vertex cover is the edge cover which seeks the smallest set of edges such that each vertex is included in one of the edges.

1.1 NP-Completeness and reducibility

In computational complexity theory, the complexity class NP-complete is a class of decision problems. A decision problem L is NP-complete if it is in the set of NP problems so that any given solution to the decision problem can be verified in polynomial time, and also in the set of NP-hard problems so that any NP problem can be converted into L by a transformation of the inputs in polynomial time. [9], [10], [11]

Although any given solution to such a problem can be verified quickly, there is no known efficient way to locate a solution in the first place; indeed, the most notable characteristic of NP-complete problems is that no fast solution to them is known. That is, the time required to solve the problem using any currently known algorithm increases very quickly as the size of the problem grows. As a result, the time required to solve even moderately sized versions of many of these problems easily reaches into the billions or trillions of years, using any amount of computing power available today. As a consequence, determining whether or not it is possible to solve these problems quickly, called the P versus NP problem, is one of the principal unsolved problems in computer science today. [12], [13]

While a method for computing the solutions to NP-complete problems using a reasonable amount of time remains undiscovered, computer scientists and programmers still frequently encounter NP-complete problems. NP-complete problems are often addressed by using approximation algorithms [14], [15], [16].

NP-complete is a subset of NP, the set of all decision problems whose solutions can be verified in polynomial time; NP may be equivalently defined as the set of decision problems that can be solved in polynomial time on a nondeterministic Turing machine. A problem p in NP is also in NPC if and only if every other problem in NP can be transformed into p in polynomial time. [17], [18].

The easiest way to prove that some new problem is NP-complete is first to prove that it is in NP, and then to reduce some known NP-complete problem to it. Therefore, it is useful to know a variety of NP-complete problems. The list below contains some well-known problems that are NP-complete when expressed as decision problems. [19],[20]

1. Knapsack problem
2. Hamiltonian path problem
3. Traveling salesman problem
4. Subset sum problem
5. Clique problem
6. Vertex cover problem
7. Graph coloring problem

1.2 Solving NP-complete problems

At present, all known algorithms for NP-complete problems require time that is super polynomial in the input size, and it is unknown whether there are any faster algorithms. [21],[22]. The following techniques can be applied to solve computational problems in general, and they often give rise to substantially faster algorithms: [23].

Approximation: Instead of searching for an optimal solution, search for an "almost" optimal one.
Randomization: Use randomness to get a faster average running time, and allow the algorithm to fail with some small probability. Note: The Monte Carlo method is not an example of an efficient algorithm, although evolutionary approaches like Genetic algorithms may be.
Restriction: By restricting the structure of the input (e.g., to planar graphs), faster algorithms are usually possible.
Parameterization: Often there are fast algorithms if certain parameters of the input are fixed.
Heuristic: An algorithm that works "reasonably well" in many cases, but for which there is no proof that it is both always fast and always produces a good result. Metaheuristic approaches are often used.

1.3 The vertex cover problem:

A vertex cover of an undirected graph G= (V, E) is a subset V'of V such that if (u, v) is an edge of G then either u $\in$ V' or v $\in$ V' (or both). The size of vertex cover is the number of vertices in it. [24], [25], [26]

The vertex cover problem is to find a vertex cover of minimum size in a given undirected graph. This problem is the optimization version of an NP-Complete decision problem.

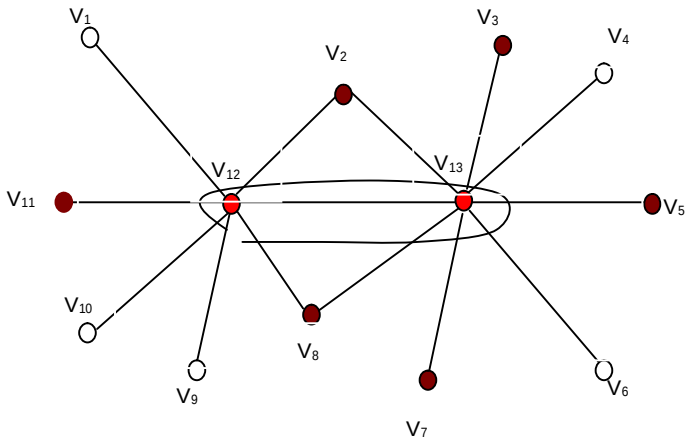

Fig. 1- The optimal cover for the given graph is C= {V_{12}, V_{13}}

2.0 PROPOSED ALGORITHM

This paper suggests a polynomial time approximation algorithm to solve the vertex cover problem. The algorithm is simple and straight to find the near optimal vertex cover to the given graph. The algorithms takes graph G (V, E) as adjacency matrix W, W= [d [i, j]] |V|× |V| / / the adjacency matrix / /. The algorithm is based on greedy approach but capable to produce the near optimal result. The algorithm is tested on various types of graphs and results given by the algorithm are accurate.

New Optimal Vertex Cover (G, W)

/ / Input: A graph G = (V, E)
/ / Output: Set C subset of V, the vertex cover.
/ / The graph is represented with adjacency matrix W (|V| × |V|)

1. D =W;
2. C= φ; / / Null set / /
3. do
4. For i =1 to |V|
5. do D[i] = $\sum_{1\le j\le |V|} d[i,j]$
6. Ki = Max {D[i]}
   $1\le i\le |V|$
7. C=C U {Vi} / / Vi is the vertex Corresponding to Ki, index i will locate Vi / /
8. For (j=1, j ≤ |V|, j++)
   Set d [i, j] =0
9. For (i=1, i ≤ |V|, i++)|
   Set d [i, j] =0
10. While (Ki ≠ 0) / / the matrix turns in to zero matrix / /
11. Return C. / / the near optimal vertex cover / /

3. THE COMPLEXITY OF THE ALGORITHM

The time complexity of the algorithms based on the following steps

1. Calculation of D[i],
2. Calculation of Ki
3. Set d [i, j] =0
4. Set d [i, j] = 0.
5. Finally, while (Ki ≠ 0)

To compute all D[i] is of order O (|V|), time to find Ki is again of order O (|V|). The step 8 and 9 both takes time of order O (|V|), all these steps are independent of each other so the total complexity of inner steps is of order (|V| +|V| + (|V| + |V|). i.e.4(|V|). while loops runs until the adjacency matrix does not turn in to Zero Matrix. The time consumed in the testing whether Ki is zero or not, takes time not more than O (|V|). So the time complexity of this algorithm in total is at most of order O (|V| (|V| +|V| + |V| + |V|)).i.e. O (|V|2). If |V| = n (says) (Number of vertices in graph G are n) then the time complexity of the algorithm is O (n2).

3.1 Applying the algorithm on a given Graph:

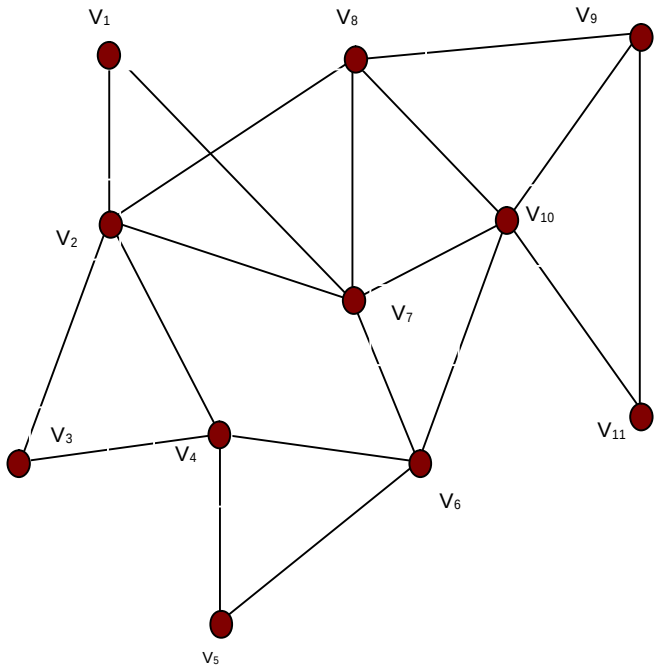

Fig.2 - The input graph

Number of nodes, n = |V| =11,
Initially C = φ
And D =

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	D[i]
V1	0	1	0	0	0	0	1	0	0	0	0	2
V2	1	0	1	1	0	0	1	1	0	0	0	5
V3	0	1	0	1	0	0	0	0	0	0	0	2
V4	0	1	1	0	1	1	0	0	0	0	0	4
V5	0	0	0	1	0	1	0	0	0	0	0	2
V6	0	0	0	1	1	0	1	0	0	1	0	4
V7	1	1	0	0	0	1	0	1	0	1	0	5
V8	0	1	0	0	0	0	1	0	1	1	0	4
V9	0	0	0	0	0	0	0	1	0	1	1	3
V10	0	0	0	0	0	1	1	1	1	0	1	5
V11	0	0	0	0	0	0	0	0	1	1	0	2

Ki = 2
C = φ U {V2} = {V2}
Setting a [2, j] = 0 for 1≤ j ≤11

and a [i, 2] = 0 for 1≤ i ≤11

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	D[i]
V1	0	0	0	0	0	0	1	0	0	0	0	1
V2	0	0	0	0	0	0	0	0	0	0	0	0
V3	0	0	0	1	0	0	0	0	0	0	0	1
V4	0	0	1	0	1	1	0	0	0	0	0	3
V5	0	0	0	1	0	1	0	0	0	0	0	2
V6	0	0	0	1	1	0	1	0	0	1	0	4
V7	1	0	0	0	0	1	0	1	0	1	0	4
V8	0	0	0	0	0	0	1	0	1	1	0	3
V9	0	0	0	0	0	0	0	1	0	1	1	3
V10	0	0	0	0	0	1	1	1	1	0	1	5
V11	0	0	0	0	0	0	0	0	1	1	0	2

Ki = 10,
C = {V2} U {V10} = {V2, V10}

Setting

a [10, j] = 0 for 1≤ j ≤11

and a [i, 10] = 0 for 1≤ i ≤11

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	D[i]
V1	0	0	0	0	0	0	1	0	0	0	0	1
V2	0	0	0	0	0	0	0	0	0	0	0	0
V3	0	0	0	1	0	0	0	0	0	0	0	1
V4	0	0	1	0	1	1	0	0	0	0	0	3
V5	0	0	0	1	0	1	0	0	0	0	0	2
V6	0	0	0	1	1	0	1	0	0	0	0	3
V7	1	0	0	0	0	1	0	1	0	0	0	3
V8	0	0	0	0	0	0	1	0	1	0	0	2
V9	0	0	0	0	0	0	0	1	0	0	1	2
V10	0	0	0	0	0	0	0	0	0	0	0	0
V11	0	0	0	0	0	0	0	0	1	0	0	1

Ki =4,

C = {V4} U {V2, V10} = {V4, V2, V10}

Setting
a [4, j] = 0 for 1≤ j ≤11
and a [i, 4] = 0 for 1≤ i ≤11

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	D[i]
V1	0	0	0	0	0	0	1	0	0	0	0	1
V2	0	0	0	0	0	0	0	0	0	0	0	0
V3	0	0	0	0	0	0	0	0	0	0	0	0
V4	0	0	0	0	0	0	0	0	0	0	0	0
V5	0	0	0	0	0	1	0	0	0	0	0	1
V6	0	0	0	0	1	0	1	0	0	0	0	2
V7	1	0	0	0	0	1	0	1	0	0	0	3
V8	0	0	0	0	0	0	1	0	1	0	0	2
V9	0	0	0	0	0	0	0	1	0	0	1	2
V10	0	0	0	0	0	0	0	0	0	0	0	0
V11	0	0	0	0	0	0	0	0	1	0	0	1

Ki =7

C = {V7} U {V4, V2, V10} = { V7, V4, V2, V10}

Setting
a [7, j] = 0 for 1≤ j ≤11

and a [i, 7] = 0 for 1≤ i ≤11

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	D[i]
V1	0	0	0	0	0	0	0	0	0	0	0	0
V2	0	0	0	0	0	0	0	0	0	0	0	0
V3	0	0	0	0	0	0	0	0	0	0	0	0
V4	0	0	0	0	0	0	0	0	0	0	0	0
V5	0	0	0	0	0	1	0	0	0	0	0	1
V6	0	0	0	0	1	0	0	0	0	0	0	1
V7	0	0	0	0	0	0	0	0	0	0	0	0
V8	0	0	0	0	0	0	0	0	1	0	0	1
V9	0	0	0	0	0	0	0	1	0	0	1	2
V10	0	0	0	0	0	0	0	0	0	0	0	0
V11	0	0	0	0	0	0	0	0	1	0	0	1

Ki =9

C = {V9} U {V7, V4, V2, V10} = {V9, V7, V4, V2, V10}

Setting
a [9, j] = 0 for 1≤ j ≤11
and a [i, 9] = 0 for 1≤ i ≤11

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	D[i]
V1	0	0	0	0	0	0	0	0	0	0	0	0
V2	0	0	0	0	0	0	0	0	0	0	0	0
V3	0	0	0	0	0	0	0	0	0	0	0	0
V4	0	0	0	0	0	0	0	0	0	0	0	0
V5	0	0	0	0	0	1	0	0	0	0	0	1
V6	0	0	0	0	1	0	0	0	0	0	0	1
V7	0	0	0	0	0	0	0	0	0	0	0	0
V8	0	0	0	0	0	0	0	0	0	0	0	0
V9	0	0	0	0	0	0	0	0	0	0	0	0
V10	0	0	0	0	0	0	0	0	0	0	0	0
V11	0	0	0	0	0	0	0	0	0	0	0	0

Ki = 5,

C = {V5} U {V9, V7, V4, V2, V10} = = {V5, V9, V7, V4, V2, V10}

Setting
a [5, j] = 0 for 1≤ j ≤11

and a [i, 5] = 0 for 1≤ i ≤11

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	D[i]
V1	0	0	0	0	0	0	0	0	0	0	0	0
V2	0	0	0	0	0	0	0	0	0	0	0	0
V3	0	0	0	0	0	0	0	0	0	0	0	0
V4	0	0	0	0	0	0	0	0	0	0	0	0
V5	0	0	0	0	0	0	0	0	0	0	0	0

V6	0	0	0	0	0	0	0	0	0	0	0	0
V7	0	0	0	0	0	0	0	0	0	0	0	0
V8	0	0	0	0	0	0	0	0	0	0	0	0
V9	0	0	0	0	0	0	0	0	0	0	0	0
V10	0	0	0	0	0	0	0	0	0	0	0	0
V11	0	0	0	0	0	0	0	0	0	0	0	0

Ki = 0 / / Zero / / Algorithm terminates
Output: So C = {V5, V9, V7, V4, V2, V10} is the near optimal cover.

4.0 APPLICATION OF THE SOLUTION: (THE WATCH TOWER CONCEPT)

High speed communication networks are the demand of time, A key challenge is to make communication network fault free and reliable, it is the responsibility of the network management to meet these challenges. Fault management is an important part of network management which is responsible for detection and identification of the network fault.
Fault management is a critical issue since fault and downtime is very costly.

Now a day the occurrence of faults in communication network are very frequent , and some time it is difficult to identify the fault quickly, also to monitor the performance of all the links in networks is vital , Vertex Cover problem suggest the concept of watch tower ,these are specific node in the communication network which are maximally connected with other nodes , with these few nodes it is possible to identify the occurrence of the fault quickly and to monitor the performance of the entire network

CONCLUSION

The greedy algorithm gives solutions better than approximation algorithm. The algorithm always makes the choice that looks best at the moment. Clever greedy algorithm always takes the vertex with the highest degree, add it to the cover set, remove it from the graph, and repeats. But the greedy heuristic cannot always find an optimal solution.
The approximation algorithms are available to solve the vertex cover problem which generally gives the near optimal solution in polynomial time. The proposed approximation algorithm provides the solution to vertex cover problem in polynomial time, though the time taken by the algorithm is a polynomial of little high of degree 2 but it is possible to reduce it further, the proposed approximation algorithm gives the optimal solution in majority of the cases but fails to provide optimal solution in some specific cases.

ACKNOWLEDGEMENT

We wish to express my sincere gratitude and cordial thanks to Prof. Kamal Ghanshala (President Graphic Era University Dehradun) for his sincere and continual encouragement in preparing this paper; thanks are also due for Dr. R.C. Joshi (Former Head- EC &CS Deptt. IIT Roorkee) (Chancellor Graphic Era University Dehradun) and Dr. Sanjay Jasola (Vice Chancellor Graphic Era Hill University Dehradun) for their continual guidance, support and helpful discussion.

REFERENCES

[1] Cook, S.A. "The complexity of theorem proving procedures". Proceedings, Third Annual ACM Symposium on the Theory of Computing, ACM, New York. p.p. 151–158. 1971.
[2] Sipser, M. "Introduction to the Theory of Computation" PWS Publishing. Sections 7.4 –7.5, pp. 248–271. ISBN 0-534-94728-X. 1997.
[3] Pemmaraju, S. and Skiena, S. "Minimum Vertex Cover Computational Discrete Mathematics Combinatorics and Graph Theory with Mathematica" Cambridge, England: Cambridge University Press, p. 317, 2003.
[4] Aho, A.V., Hopcroft, J.E. and Ullman, J.D. "The Design and Analysis of Computer Algorithms" Addison –Wesley, 1975.
[5] George K. "A better approximation ratio for the vertex cover problem." In ICAL 2005, volume 3580 of LNCS, p.p 1043–1050, Lisboa, Portugal, June 2005. Springer-Verlag Berlin Heidelberg.
[6] Clarkson, K. "A modification to greedy algorithm for Vertex Cover Problem" IP: Vol 16: p.p 23-25, 1983.
[7] Garey, M. R. and Johnson, D. S. "Computers and Intractability: A Guide to the Theory of NPCompleteness." W. H. Freeman, 1979.
[8] Hartmann, A. K., & Rieger, H. (Eds.) (2004). "New optimization algorithms in physics". Weinheim: Wiley-VCH.
[9] Dorits S. Hochbaum" Approximation algorithm for NP-Hard problems", 2002.
[10] Levin, L. A. "Average case complete problems" SIAM J. Comput., 15(1) p.286, 1986.
[11] Sipser, M. "The history and status of the P versus NP question" In Proc. ACM STOC, p.p. 603–618, 1992.
[12] Kozen, D. C. "The Design and Analysis of Algorithms" Springer, USA, ISBN-0-367-97667-6, ISBN-3-540-97687-6, 1991.
[13] Levitin, A. "Introduction to Design and Analysis of Algorithm" Pearson Education Inc. and Dorling Kindersley Publishing ISBN 81-7758-835-4, New Delhi.
[14] Thomas C. and O'Connell. "Fundamental Problems in Computing, chapter: A survey of graph algorithms under extended streaming models of computation." p.p 455–476. Springer Science Business Media, 2009.
[15] Papadimitriou, C. and Yannakakis, M. "Optimization, approximation, and complexity Classes", Journal of Computer and System Sciences 43, p.p, 425-440 (1991).
[16] Jörg,F. and Martin, G. "Parameterized Complexity", Theory. Springer. ISBN 978-3-540-29952-3., 2006.
[17] Richard, M. K. "Reducibility among combinatorial problems. Complexity of Computer Computations," 1972.
[18] Corman, H.T., Leiserson, E.C., Rivest,L.R., and Stein, C. "Intro-

duction to Algorithms" 2nd, Prentice Hall of India private Ltd, New Delhi , 2004.

[19] Vazirani, V., V. "Approximation Algorithms". Springer-Verlag. ISBN 3-540-65367-8, 2001.

[20] Bollobas. "Random Graphs", Cambridge University Press, Cambridge U.K, 2nd edition, 2001.

[21] Bar, R. –Yehuda and Seven "A linear time approximation algorithm for weighted vertex cover problem", J.Algorithms, Vol. 2: p.p 198-203, 1981.

[22] Peliken, M. and Bayesian, H. "Optimization algorithm: towards a new generation of evolutionary algorithms" Springer –Verlag p.p.102-160. 2005

[23] Weight. M.and Hartmann, A. K. (2000a). "Minimal vertex cover on finite-connectivity random graphs – A hard-sphere lattice-gas picture". Phys. Rev. E, 63, 056127.

[24] Garey, M.R. and D.S. Johnson. "Computers and Intractability". W.H. Freeman, 1979.

[25] Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. Determining computational complexity from characteristic phase transitions. Nature, 400, 133. (1999).

[26] Weigt, M., & Hartmann, A. K. (2000b). "The number guards needed by a museum – a phase transition in vertex covering of random graphs". Phys. Rev. Lett., 84, 6118.