A Method to Construct Counterexamples for Greedy Algorithms

Jagadish M
Dept. of Computer Science and Engg.
Indian Institute of Technology Bombay
Mumbai 400076, India
jagadish@cse.iitb.ac.in

Sridhar Iyer
Dept. of Computer Science and Engg.
Indian Institute of Technology Bombay
Mumbai 400076, India
sri@iitb.ac.in

ABSTRACT

Problem solving is the focus of any algorithm design course. Being able to construct counterexamples is a critical part of the problem solving process. We address the following question: Is there a *teachable* approach to construct counterexamples? We present a technique, called the *Anchor Method*, that could prove useful in constructing counterexamples for greedy algorithms. The basic idea of the method is to get insight about the structure of a counterexample by weakening the greedy algorithm. We show the applicability of the method by constructing counterexamples for some classic problems in graph theory. We also investigate the teachability of the technique through experiments.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms

Algorithms

Keywords

Counterexamples, Greedy algorithms, Problem-solving

1. INTRODUCTION

In the design of algorithms, the discovery of a counterexample can help in discarding a wrong approach. Counterexamples developed for failed approaches usually serve as test cases for new approaches and may also yield insight into the problem structure.

Constructing counterexamples can be hard. Early research in cognition shows that people are more likely to look for examples to confirm a hypothesis than to disprove it[6]. For example, it is common for students to come up with an intuitive (but wrong) algorithm and use input cases only to assure themselves of its correctness, instead of finding a rigorous proof or searching for a counterexample[3].

ITiCSE'12, July 3–5, 2012, Haifa, Israel.

Greedy, divide-and-conquer, recursion, dynamic programming, reduction, etc. are some of the prominent algorithm design techniques that are taught at undergraduate level. Counterexamples often crop up in the context of greedy algorithms. For example, every third exercise problem in the chapter on greedy algorithms in the Klienberg-Tardos textbook[5] requests for a proof or a counterexample. "Exchange argument" is often helpful in proving the correctness of a greedy algorithm. To the best of our knowledge, there is no method aimed at constructing a counterexample to disprove a greedy approach. Students simply rely on their intuition or search for counterexamples by trying small cases.

We present a technique, called the *Anchor method*, that could prove useful in constructing counterexamples for greedy algorithms. We limit the scope of our work to simple greedy approaches since most of the problem solver's first plans of attack tend to be greedy in nature. We also limit the scope of our problems to graph theoretic problems on unweighted graphs, since many problems in computer science map to this class[8].

In the next section, we introduce the notion of *weak algorithm* and define three types of counterexamples: *weak*, *plausible* and *definitive*. We will use the Maximum Independent Set (MIS) problem as a running example to clarify these definitions and also illustrate the working of the anchor method (section 3). In section 4, we use the anchor method to construct counterexamples for a few well-known problems in graph theory–this shows the general applicability of our method. In section 5, we address questions related to the teachability of the method.

2. DEFINITIONS

Weak Algorithm. The greedy algorithm works by 'making the choice that looks best at the moment' [5]. We define a weak algorithm to be an algorithm that works by making an *arbitrary* choice at each moment.

Weak Counterexample. A weak counterexample is an instance on which the weak algorithm fails assuming the worst possible execution or, in other words, when choices are made in an adversarial manner (e.g. Fig. 4).

Plausible and Definitive Counterexample. If more than one choice looks best at a given step, the greedy algorithm makes a choice non-deterministically. Hence, the algorithm could have multiple execution paths for a given input instance. We call an input instance a *plausible counterexample* if one of the execution paths leads to a wrong answer. Similarly, an input instance is called a *definitive*

counterexample if every possible path of execution leads to a wrong answer.

Maximum Independent Set Problem

A *graph* G consists of a set V of *vertices* and a set E of pairs of vertices called *edges*. For an edge $e = (u, v)$, we say e is *incident* on vertex u and v; vertices u and v are also the *endpoints* of e. The pair of vertices in an edge is unordered so $e = (u, v)$ is the same as $e = (v, u)$. The *degree* of a vertex is the number of edges incident upon it.

Problem. Let $G = (V, E)$ be an graph. The maximum independent set problem (MIS) is to find the largest vertex set $S \subseteq V$ such that no two vertices in S have an edge between them in E. See Fig. 1.

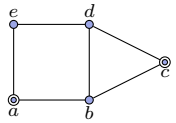


Figure 1: In the graph shown, $S = \{a, c\}$ is a maximum independent set.

Consider the following greedy approach to solve the MIS problem:

Algorithm. When a vertex is picked, we are prohibited from picking its neighbors. Since we want to pick as many vertices as possible, we start by picking vertices with low degree[7].

Algorithm 2.1: MIS(G)

Input: Graph G

Output: An independent set S from graph G

Initially the set S is empty.
1 : Sort the vertices according to their degree.
Let V_s be the sorted sequence.
2 : **for each** vertex v in V_s
if vertex v does not have a neighbor in S
add vertex v to set S.
3 : Report S as the answer

Consider the input graph in Fig. 2: a cycle of length 6 (C_6). The size of the maximum independent set in this graph is 3 (set $\{a, c, e\}$ for example).

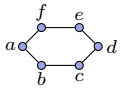


Figure 2: Cycle C_6 has 3 independent vertices.

As all the degrees of vertices are same, the algorithm could sort them in any order. If $V_s = [a, b, c, d, e, f]$, the algorithm returns $\{a, c, e\}$ as the maximum independent set. However, if $V_s = [a, d, f, e, b, c]$, the algorithm returns $\{a, d\}$ as the maximum independent set (incorrectly). Hence the input graph C_6 is a plausible counterexample for the given greedy algorithm.

In general, we can convert a plausible counterexample into a definitive counterexample with some modification. In the above example, we want the algorithm to pick vertices a and d before others. We can ensure that by adding the edges (b, e) and (c, f) as shown in Fig. 3.

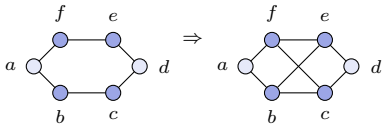


Figure 3: Converting a plausible counterexample to a definitive counterexample. The size of the MIS in the second graph is 3 ($\{a, c, e\}$), but the greedy algorithm 2.1 always returns a set of size 2 ($\{a, d\}$).

Although a definitive counterexample is more desirable, providing a plausible counterexample is a legitimate way of disproving a greedy algorithm.

3. WORKING OF THE ANCHOR METHOD

Our objective is to construct a counterexample to a given greedy algorithm. The anchor method follows a basic principle of problem-solving: solve an easier problem first. Instead of constructing a greedy counterexample directly, we first construct a counterexample for the weak algorithm. We call this a *weak counterexample*. Recall that a weak algorithm is obtained from the greedy algorithm by replacing the greedy choice with an arbitrary choice. The assumption is that the weak counterexample gives insight into the structure of the greedy counterexample. Our method consists of two steps:

Step 1 **Build an anchor** Use the weak counterexample to obtain an *anchor*. The anchor is a skeleton that captures the essence of a possible greedy counterexample. It serves as a starting point for constructing a counterexample to the greedy algorithm.

Step 2 **Add an auxiliary structure** Attach an *auxiliary structure* to the anchor to obtain a complete counterexample. The purpose of the auxiliary structure is to ensure that the greedy algorithm essentially mimics the weak algorithm (and hence fails in the same way). Auxiliary structures usually have 'extreme' or 'common' structures.

We apply the anchor method to construct a counterexample to Alg. 2.1 for the MIS problem. This algorithm picks the vertices in increasing order of degree, so the corresponding weak algorithm picks vertices in arbitrary order.

3.1 Maximum Independent Set: An illustrative example

Building an Anchor. Consider the weak algorithm that picks the vertices in arbitrary order to include in S. A counterexample to this algorithm is shown in Fig. 4.

The weak algorithm could pick the central vertex first and get an independent set of size 1, whereas the maximum independent set has size 3 ($\{a, b, c\}$). So this is our weak counterexample.

In the weak counterexample, the central vertex has the highest degree. We know that the greedy algorithm would pick the central vertex in the above graph first if the central

Figure 4: A counterexample to the weak algorithm for the MIS problem. A star graph with three outer vertices.

vertex had the *lowest* degree *somehow*. The anchor just captures this intuition (Fig. 5).

Notation. We use symbols ● and ○ to indicate a vertex we *wish* were of high degree and low degree, respectively.

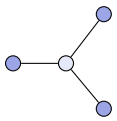


Figure 5: The anchor obtained from the weak counterexample.

Adding the auxiliary structure. Each high degree vertex needs to be connected to at least 3 vertices in order to exceed the low degree central vertex. Let K_n denote a complete graph with n vertices. Attaching the auxiliary graph K_3 gives a counterexample as shown in Fig. 6.

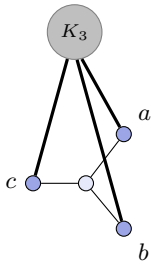


Figure 6: A counterexample to Alg. 2.1 for the MIS problem. A dark edge indicates that there are edges from the high degree vertex to all the three vertices in K_3. The central vertex has degree 3. Vertices a, b and c have degree 4 each. Each vertex in K_3 has degree 5. So the greedy algorithm picks the central vertex first and then picks one vertex from K_3. The maximum independent set is actually $\{a, b, c\}$. Hence, the greedy algorithm picks 2 vertices, while the optimal algorithm picks 3.

Approximation ratio. The approximation ratio is a measure of how good a counterexample is. It is defined as follows: Suppose $\mathcal{I}$ is an input to an algorithm $\mathcal{A}$. Let OPT be the value of the optimal solution to $\mathcal{I}$ and ALG be the value returned by the algorithm $\mathcal{A}$. The approximation ratio of the input $\mathcal{I}$ is defined the ratio of OPT to ALG, for maximization problems. For minimization problems, it is the ratio of ALG to OPT. For example, for the input shown in Fig. 6 OPT=3 and ALG=2, so the approximation ratio is 1.5.

Note that we can pick any star graph with $n(\geq 3)$ outer vertices as our weak counterexample. Attaching an auxiliary

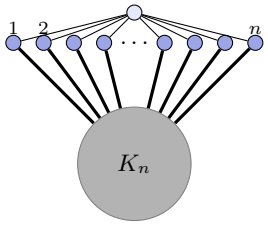


Figure 7: A general counterexample to Alg. 2.1 for the MIS problem with $O(n)$ approximation ratio. The greedy algorithm picks 2 independent vertices, while the optimal algorithm picks n vertices.

graph K_n to the corresponding anchor gives us a *general* counterexample (Fig. 7) with the best approximation ratio.

We may have to try a few weak counterexamples before hitting upon the one which extends to a greedy counterexample.

4. ADDITIONAL EXAMPLES

We show the applicability of the anchor method on many classic problems from graph theory. Most of the problem descriptions along with their applications can be found in standard texts like [2, 9].

4.1 Matching

A set of edges $M \subseteq E$ is called a *matching* if every vertex in G is an endpoint of at most one edge in M. A vertex is said to be *matched* if it is an endpoint of some edge in M; if not the vertex is said to be *unmatched*.

Problem Definition. Let $G = (V, E)$ be a graph. The maximum matching problem (MM) is to find a matching M of maximum size in G. An example is shown in Fig. 8.

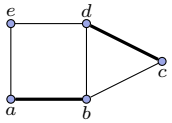


Figure 8: In the graph above, the dark edges form a matching of size 2 (which is also the maximum size).

Algorithm. A vertex with degree one has to necessarily be matched to its neighbor if it has to participate in the matching. In general, a vertex with fewer neighbors has fewer choices and is therefore more critical.

Notation. Let v_{min} denote the vertex with minimum non-zero degree in graph G and u_{min} denote the vertex with minimum degree adjacent to v_{min}.

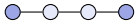
The greedy algorithm is to pick the edge (v_{min}, u_{min}) and then remove the vertices v_{min} and u_{min} from G. Repeat the process on the resulting graph until an edge can be picked.

Counterexample

Building an Anchor. Consider the weaker algorithm that picks the edges in any order to include them in the matching. A path of four nodes is the simplest instance on which the weak algorithm fails (Fig. 9). The weak algorithm could pick just the middle edge, while the optimal algorithm picks the first and the last edge.

The anchor follows from the above weak counterexample:

Figure 9: A counterexample to the weak algorithm for the matching problem.



Adding the auxiliary structure. We attach K_2 to each of the end vertices. This gives a plausible counterexample–the greedy could pick a matching of size 3 instead of 4. Attaching K_4 gives a definitive counterexample.

Figure 10: Plausible and definitive counterexamples to the greedy algorithm for the matching problem.

4.2 Maximum leaf spanning tree

Problem Definition. Given a graph $G = (V, E)$, the maximum leaf spanning tree problem(MLS) is to find a spanning tree T in G that has maximum number of leaves; where leaves are vertices with degree 1. An example is shown in Fig. 11.

Algorithm. We would like to grow the tree along vertices that branch out well. One heuristic is to grow the tree along a vertex which has the maximum number of neighbors not in the tree[4].

Notation. For a vertex v in tree T, let $N'(v)$ denote the set of neighbors of v not in T.

The greedy algorithm initializes the tree T with the maximum degree vertex and then grows the tree along that vertex u for which the size of $N'(u)$ is maximized.

Counterexample

Building an Anchor. In the above algorithm, we grow the tree T along the vertex that has maximum number of neighbors outside the tree. Consider the weaker algorithm that does not exercise this choice and instead grows the tree along any arbitrary vertex.

An instance where the weak algorithm could perform poorly is shown in Fig.12. The optimal algorithm would pick the vertex with the highest degree r and obtain a spanning tree having $|V|-1$ leaves. But if the weak algorithm picks vertex l first, it could end up with a spanning tree having only two leaves (as indicated with dark background).

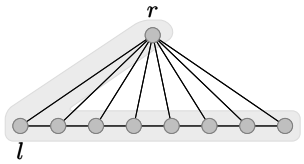
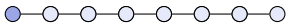


Figure 12: A counterexample to the weak algorithm for the MLS problem.

Notice that the key to designing the weak counterexample was to make the algorithm select a long path of vertices which should have been selected as leaves. Can we mimic that behavior for the greedy algorithm? We make the greedy algorithm start from the first node by increasing its degree.



Adding the auxiliary structure. We need to make the vertices of this path as leaves of some tree. We need to attach an auxiliary structure such that the vertices on the path become leaves. The auxiliary structure cannot have a vertex with degree greater than 3 since we have forced the greedy algorithm to pick the the leftmost vertex in the anchor first. The complete binary tree serves this purpose. The counterexample is shown in Fig. 13.

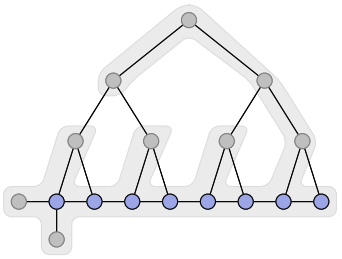


Figure 13: A plausible counterexample to the greedy algorithm for the Max-leaf Spanning Tree problem.

4.3 Minimum Steiner tree

Problem Definition. Given a graph $G = (V, E)$ and a set of *terminal* vertices $P \subseteq V$, the minimum Steiner tree problem (STP) is to find a tree T in G such that the following two conditions are satisfied:

1. Tree T contains all terminal vertices P.

2. The number of edges in T is minimized.

An example is shown in Fig. 14.

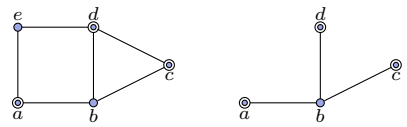


Figure 14: The input graph G with terminals $P = \{a, c, d\}$ has a minimum Steiner tree having three edges. Note that the Steiner tree can contain vertices not in set P.

Algorithm. Notice that the Steiner tree must connect all the terminals using the least number of edges. Since any two terminals must be connected, we try to grow the tree by connecting two vertices along their shortest path.

Definition. For a tree $T \in G$ and a vertex $v \in G$, a $v \rightsquigarrow T$ *connecting path* is the shortest path that connects vertex v to some vertex in T.

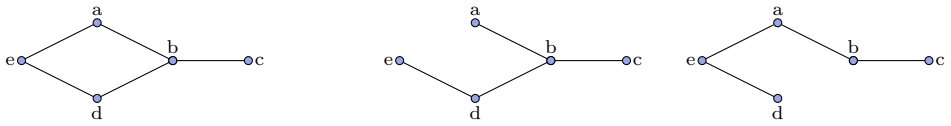


Figure 11: Two spanning trees for the graph given on the left: one with 3 leaves and the other with 2 leaves.

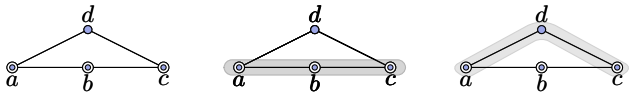


Figure 15: A weak counterexample for Steiner tree problem

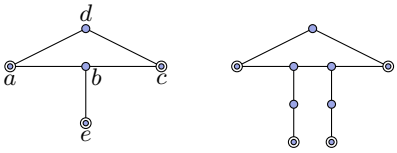


Figure 16: Plausible and definitive counterexamples to Alg. 4.1 for the Steiner tree problem.

Algorithm 4.1: STP(G)

Input: Graph G and a terminal set P

Output: A minimum Steiner tree of G

Initially, the tree T contains only the shortest path amongst all pairs of terminals in the set P.
repeat
1 : Let u be the terminal not in T that has the shortest connecting path to T.
2 : Add $u \rightsquigarrow T$ connecting path to T.
until the tree T contains all the terminals in the set P.
Report tree T as the Steiner tree for the graph G and terminals P.

Counterexample

Building an Anchor. Note that the greedy algorithm starts by picking the shortest path between all pairs of terminals. We weaken the algorithm slightly by saying the tree could be initialized with the shortest path between any two terminals.

Let us try to build a counterexample for the weak algorithm. It is possible to construct a counterexample having 3 terminals as shown in Fig 15. This is the simplest possible input since no counterexample is possible with only two terminals.

The optimal Steiner tree is the path containing 3 terminals. The weak algorithm could initialize tree T by picking the shortest path between a and c going via the non-terminal vertex d. This would result in a Steiner tree containing 3 edges. This serves as our anchor.

Adding the auxiliary structure. If the greedy algorithm has to take the path adc first, we cannot afford to have distance of 1 between any pair of terminals. We add an edge to move away vertex c so as to ensure all the shortest paths between terminals are 2. This is a plausible counterexample. We can extend the same idea to obtain a definitive counterexample as shown in Fig. 16.

5. EXPERIMENTS

We conducted two experiments to assess the educational value of the anchor method. Note that the anchor method assumes that the weak counterexample is easier to construct than the greedy counterexample. Though it seems like a reasonable assumption, we conducted an in-class experiment to test this assumption (Experiment 1).

The second experiment was of qualitative nature. The purpose was to find the strategies that students employ, common mistakes they make, while constructing counterexamples and to identify the difficulties they face in applying the anchor method. The small scale of this experiment does not allow us to establish that the anchor method is actually teachable, but going through the exercise has been helpful in identifying a few key aspects related to problem-solving. We describe the setup of two experiments below and then address questions related to the teachability of the method.

Experiment 1. Thirty eight students who were enrolled in a minor course on "Data structures and algorithms" took the test which had two questions on the MIS problem. The class consisted of top 5% second year undergraduate students (CGPA-wise) from different departments from a top ranked engineering institute. The first and the second question asked them to construct a counterexample for the weak and greedy algorithm for the MIS problem, respectively. The summary of results are given in Table 1.

Null Hypothesis. The number of students who are able to construct a weak counterexample (WCE) is same as the number of students who can construct a greedy counterexample (GCE) for the MIS problem.

	WCE	**GCE**
No of correct responses	37/38	19/38
Average time taken by those who solved the question	4 min	20 min

Table 1: Summary of students' responses

The result of the experiment rejects our null hypothesis with effect size (as defined in [1]) greater than 3.8. Hence, we conclude that students find it easier to construct the weak counterexample than the greedy counterexample for the MIS problem.

Experiment 2. A qualitative experiment with three graduate students: A (PhD student), B (Masters student) and C (PhD student). The purpose of the experiment was to observe how students approach problems in a detailed manner. A and B were part of the experimental group that was taught the anchor method after the pre-test, while C was the control subject.

Design. Pre-test and Post-test each followed by an interview. Pre-test contained Max-Independent Set and

STEINER TREE problems. Post-test contained MATCHING and MAX-LEAF problems. After the pre-test, Students A and B were taught the anchor method by using the pre-test problems as illustrative examples. Student C was not taught the anchor method but was simply given the solutions to pre-test problems.

Questions on teachability. We answer questions related to the teachability of the method based on the observations from the above experiments.

How difficult is it for a student to come up with greedy counterexamples?

In Exp. 1, only 19/38 gave the greedy counterexample for the MAXINDEPENDENT set problem, while 37/38 students were able to design a weak counterexample. In Exp. 2 pre-test, only student A was able to solve the the MAXINDEPENDENT set problem. The Steiner tree problem was solved by students A and C.

However, none of the students in both the experiments constructed a counterexample with $O(n)$ approximation ratio for the MIS problem. This shows that it is generally difficult for students to construct counterexamples for greedy algorithms, especially the ones with good approximation ratios.

What heuristics do students commonly apply while constructing counterexamples?

These observations are based on Exp. 2:

- *Force the algorithm to make the wrong choice first.* This by far seems to be the most popular heuristic. For the MIS problem, all three students wrote a star first and tried to tweak the example such that picking the central vertex leads to a bad choice. This heuristic was unsuccessful for the MIS problem.

- *Weaken the input.* For example, instead of building a Steiner tree with unweighted vertices student A first built a counterexample with weighted graph and then converted it into an unweighted graph.

- *Work backwards from the solution.* Student B and C reported having tried this heuristic on MIS and Matching problem, respectively.

In Exp. 2, were students in the experimental group able to apply the anchor method in the post-test?

Note that applying the anchor method consists of two steps: coming up with the weak counterexample and attaching a suitable auxiliary structure. Students A and B were able to construct weak counterexamples for MATCHING and MAX-LEAF problems. Both the students could construct the counterexample for the MATCHING problem by attaching a complete graph K_4. However, only student A was able to attach the auxiliary structure for MAX-LEAF problem. Student B was not able to think of using a complete binary tree as an auxiliary structure. This suggests that step 2 is harder than step 1 and requires some practice.

How did student C fare in the post-test? Student C was unable to solve both the MATCHING and the MAX-LEAF problems. During the post-test C looked at solutions to the pre-test problems to see if he could construct a counterexample along the same lines. He mostly tried the heuristics listed above and some ad-hoc methods before giving up. This shows that it is not easy to *intuit* the anchor method just by looking at the solutions.

6. CONCLUSION

A couple of points in favor of the anchor method are as follows. Techniques like greedy, divide-and-conquer, dynamic programming, etc. are quite general and hence the student does not usually know *when* a particular method is applicable. Unlike these techniques, the anchor method is tailor-made for constructing counterexamples for greedy algorithms, so there is a greater chance of recall. In the examples considered, our approach usually led to a general counterexample with good approximation ratio. No student gave a general counterexample for the greedy algorithm in our experiments.

We observed that students sometimes fail to acknowledge plausible counterexamples as legitimate counterexamples and seek to get a definitive counterexample directly. Hence, we recommend that the distinction between a plausible and a definitive counterexample be made clear to students.

The second step of attaching an appropriate auxiliary structure is still descriptive. What constitutes an auxiliary structure is problem-specific. For the examples discussed, the auxiliary structures turned out to be simple (complete graphs, complete binary tree, etc). But other problems may require more sophisticated auxiliary structures. We believe that a technique to identify auxiliary structures would strengthen the anchor method.

7. ACKNOWLEDGMENTS

We thank Soumitra Pal for suggesting improvements to the presentation of the paper.

8. REFERENCES

[1] L. Cohen, L. Manion, and K. Morrison. *Research Methods in Education*. Routledge, 6 edition, 2007.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to NP-Completeness*. Freeman, San Francisco, CA, USA, 1979.

[3] D. Ginat. The greedy trap and learning from mistakes. *SIGCSE Bull.*, 35:11–15, January 2003.

[4] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20:374–387, 1996.

[5] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison Wesley, second edition, 2006.

[6] J. V. Oakhill and P. N. Johnson-Laird. Rationality, memory and the search for counterexamples. *Cognition*, 20(1):79–94, 1985.

[7] S. Sakai, M. Togasaki, and K. Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Appl. Math.*, 126:313–322, March 2003.

[8] R. Sedgewick and K. Wayne. Undirected graphs applications. URL (accessed 2011-08-17). `http://www.cs.princeton.edu/~rs/AlgsDS07/11UndirectedGraphs.pdf`.

[9] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, September 2000.