

This article was downloaded by: [CDL Journals Account]

On: 28 March 2009

Access details: Access Details: [subscription number 785022369]

Publisher Routledge

Informa Ltd Registered in England and Wales Registered Number: 1072954 Registered office: Mortimer House, 37-41 Mortimer Street, London W1T 3JH, UK



Computer Science Education

Publication details, including instructions for authors and subscription information:

<http://www.informaworld.com/smpp/title-content=t713734307>

A puzzle-based seminar for computer engineering freshmen

Behrooz Parhami^a

^a Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA

Online Publication Date: 01 December 2008

To cite this Article Parhami, Behrooz(2008)'A puzzle-based seminar for computer engineering freshmen',Computer Science Education,18:4,261 — 277

To link to this Article: DOI: 10.1080/08993400802594089

URL: <http://dx.doi.org/10.1080/08993400802594089>

PLEASE SCROLL DOWN FOR ARTICLE

Full terms and conditions of use: <http://www.informaworld.com/terms-and-conditions-of-access.pdf>

This article may be used for research, teaching and private study purposes. Any substantial or systematic reproduction, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The accuracy of any instructions, formulae and drug doses should be independently verified with primary sources. The publisher shall not be liable for any loss, actions, claims, proceedings, demand or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.

A puzzle-based seminar for computer engineering freshmen¹

Behrooz Parhami*

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA

We observe that recruitment efforts aimed at alleviating the shortage of skilled workforce in computer engineering must be augmented with strategies for retaining and motivating the students after they have enrolled in our educational programmes. At the University of California, Santa Barbara, we have taken a first step in this direction by offering a required freshman seminar entitled “Ten Puzzling Problems in Computer Engineering”. This one-unit pass/not-pass gateway course, which is graded based solely on attendance, introduces our students to some of the most challenging problems faced by computer engineers in their daily professional endeavors and at the frontiers of research. To accomplish this feat in a manner that is both understandable and appealing to freshmen, the problems are related to popular mathematical and logical puzzles. Each 1-hour class session begins by introducing the students to puzzles of a particular kind and letting them participate in formulating solutions. Historical context, background, and general solution methods for the puzzles are then discussed by the instructor, who finally proceeds to demonstrate how the puzzles and their solution strategies are related to real technical challenges in computer engineering. The new course, which has been offered twice already, is supported by a website containing complete lecture slides, class handouts, and reference information.

Keywords: computer science and engineering education; freshman seminar; problem complexity; problem-solving method; puzzle; puzzling problem

Introduction

Freshman seminars are being offered by many universities. Reasons for offering these seminars are various. Some are aimed at helping students with the transition from high school to college (Appalachian State University, 2008). Others strive to create “the excitement and challenge of working in a small setting with a professor and fellow students on a topic of special interest” (Princeton University, 2008). A third category aims at introducing college freshmen to general challenges of college-level learning, resources available to students, and important study skills (O’Connor & Williams, 2007). Still others aim to present a sampling of milestones, achievements, and societal impacts of computing and information sciences (University of California Santa Barbara, 2006).

Anecdotal evidence on the usefulness and effectiveness of freshman seminars has led to a proliferation of such offerings in many institutions. In this paper, we report on the efforts and designs for a fifth kind of freshman seminar within the Computer Engineering Program at the

*Email: parhami@ece.ucsb.edu

University of California, Santa Barbara (UCSB): one meant to introduce the students to the most challenging problems faced in their selected field of study by practicing engineers who design state-of-the-art products or processes and by researchers at the forefront of technology (Parhami, 2008). Lecture topics are chosen such that they can be related to interesting mathematical or logical puzzles, which lead off the discussion in each class session.

A complete syllabus for this course, along with teaching material for each of the 10 lectures, is available at the course's website (Parhami, 2008). The teaching material for each lecture comprises complete presentation files in PowerPoint and PDF formats, and URLs of websites where additional information about the puzzles can be found. The PowerPoint presentations contain a great deal of animation for revealing information gradually or overlaying different components of a diagram. Because these animations are lost in the PDF files, it is recommended that the PowerPoint files be used whenever possible. Online course resources also include a four-slide handout (printable on two sides of a single sheet, with two slides per page) that allows the students to participate in solving puzzles in the early part of the lecture and makes it possible for them to take away some of the more challenging puzzles for further enjoyment or sharing with friends and family outside the classroom.

The need for a gateway course

Much has been written about the shortage of skilled workforce in information technology and the concomitant need to attract more students to computer science and engineering programmes (Akbulut & Looney, 2007). The shortage has been attributed to the perceived impact of the dot-com demise, company consolidations, and off-shoring. An anticipated wave of retirements might worsen this already dire situation in the USA (National Association of State Chief Information Officers, 2007). Similarly acute skill shortages are being faced in Europe (Jain, 2007), Asia, and Australia. We maintain that attracting students to computer science and engineering programmes, while necessary and helpful, counteracts only one aspect of the problem. Retaining and motivating students once they have chosen a computing major are other key aspects. Attention to student retention and motivation has unfortunately been lacking in the curricula recommendations by professional organisations such as IEEE Computer Society and ACM (IEEE-CS/ACM Joint Task Force on Computing Curricula, 2002, 2006).

As a case in point, aggregate data for computer engineering (CE) majors at UCSB suggest that only about one-third of entering freshmen graduate in their major and that nearly half have already left the CE major by the start of their third year (see Figure 1). Part of this loss is attributable to the fact that freshmen are admitted to the CE pre-major, and they are required to satisfy certain unit and grade-point requirements to advance to the CE-major status. Additionally, some attrition due to students discovering that their talents and interests lie in other disciplines (and thus opting for a change of major) is quite natural and does not necessarily constitute a loss for the university or society. However, a high attrition rate, combined with the challenge of attracting top-notch applicants to CE, does not bode well for our discipline. A greater retention rate will improve the quantity and quality of our graduates much more effectively than simply admitting more students, as the latter approach would require digging deeper into the applicant pool.

Because of the need to master foundational and basic-science notions before dealing with their real-world applications, college students in engineering and technology usually do not come in full contact with their chosen discipline until the third year of their studies. This situation is detrimental to keeping students motivated. Engineering freshmen simply see their early college experience in the same vein as their high school coursework, given

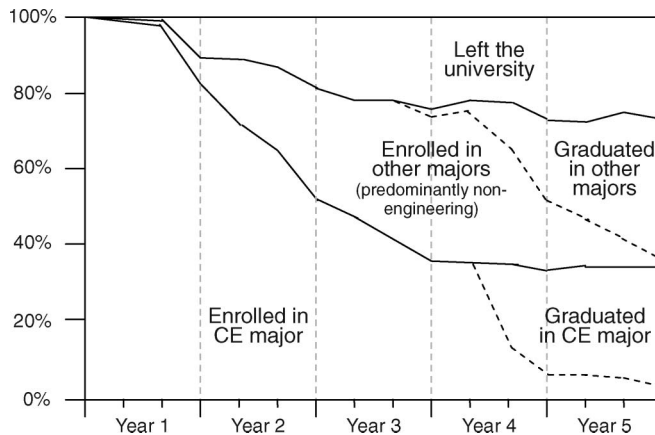


Figure 1. Retention of entering freshmen in computer engineering by academic quarter (courtesy of UCSB Office of Institutional Research and Planning).

that they are not presented with a context relating mathematics, physics, or even programming courses to practical problems in their chosen fields of study. Many students in computer science and engineering excel in mathematics, physics, and other such subjects, but are averse to studying them in isolation; after all, they could have chosen a basic science major if they were so inclined. Students who are less at ease with these foundational topics suffer even more.

There is, therefore, a need to offer the entering freshmen a glimpse of the types of problems that they will be facing as practicing engineers or as scientists at the frontiers of research. Of course, one cannot discuss such problems in highly technical terms, given that most freshmen would not be able to grasp the underlying ideas. This is where puzzles enter the picture, allowing us to introduce the topics in a comprehensible and entertaining way.

The role of puzzles in instruction

Many engineering problems are puzzlelike. Pieces of the puzzle are provided to engineers in the form of user/customer requirements, technological constraints, professional or industrial codes, and market realities. The engineer must then craft a product or process that either meets all these (often conflicting) demands or else provides partial solutions, with clear justification of the tradeoffs made when meeting all of the specifications is not possible. Even though all engineers deal with puzzlelike undertakings, computer engineers seem to face a much greater share of such problems. Puzzling problems are, of course, plentiful in the research arena, regardless of the discipline.

So, what exactly is a puzzling problem? In a very interesting resource for teaching of physics, the authors use the term “puzzling problem” as an antithesis to problems that “can be solved only through long, complex calculations, which tend to be mechanical and boring, and often drudgery for students” (Gnädig, Honyek & Riley, 2001). According to this view, a puzzling problem may ask, “How high could the tallest mountain on Mars be?” Others have used the term to refer to trick questions, or riddles, such as “Is it against the law for a man to marry his widow’s sister?” or “How much dirt is there in a 3-meter by 3-meter square hole that is 2 meters deep?”

In our usage here, a puzzling problem is either an innocent-looking problem that reveals its depth and degree of difficulty when one begins to formulate a solution for it

(e.g., rotate faces of a Rubik's cube until each of the six faces is single-coloured), or a tough-looking problem that is readily tamed with the correct insight (e.g., draw four straight lines that connect all nine points in a 3×3 grid, without lifting your pen). The use of puzzles has a long tradition in teaching of mathematics (see puzzle collections on the website of the Center for Innovation in Mathematics Teaching. <http://www.cimt.plymouth.ac.uk/resources/puzzles/default.htm>; and Parker, 1955). The author's own experience, and relative success, in using puzzles to teach mathematics topics to elementary school students (see "Math + Fun!", a permanent website containing a number of presentations on mathematics and technology topics: <http://www.ece.ucsb.edu/~parhami/math-plus-fun.htm>) was instrumental in imagining and designing the freshman seminar course described in this paper. Puzzles have also been advocated as teaching aids in other disciplines: computer science (see *Computer Science Unplugged*, learning activities based on games and puzzles for primary-aged children: <http://csunplugged.org/index.php/en/>; Hill, Ray, Blair, & Carver, 2003; Levitin, 2003; Levitin & Papalaskari, 2002), operations research (Müller-Merbach, 1975), and biology (Franklin, Peat & Lewis, 2003), to name a few examples. Mathematical and logical puzzles have also been used as tools for honing students' problem-solving skills, regardless of the field of study (Averback, 1980).

The puzzling nature of CE problems is best captured by two recent articles. In an informative article, Lauren Aaronson (2006) demonstrates the relationship between the immensely popular Sudoku puzzles and the NP-complete class of computational problems which arise in practically important application domains such as scheduling, network routing, and gene sequencing. In another insightful article, Brian Hayes (2007) deals with mechanisms for rearranging train cars in railroad yards, showing how these mechanisms relate to stacks, queues, and other data structures and associated algorithms. These two sources were used as bases for two of the 10 lectures in the course described herein (see Appendices 7 and 9).

The 10 course lectures

In Appendices 1–10, the 10 lectures of the course are discussed briefly. For each lecture, a figure is supplied which contains example puzzles as well as one key computer engineering idea related to the puzzles. Appendix headings match the lecture titles, which are also listed in Table 1 alongside the puzzles used, for ease of reference.

The format of lectures is as follows. At the beginning of each class session, the instructor introduces a simple puzzle and asks the students to try to solve it. Students are

Table 1. List of discussion topics and the associated puzzles.

	Lecture title	Lead puzzle
Lecture 1	Easy, hard, impossible!	Collatz's conjecture
Lecture 2	Placement and routing	Houses and utilities
Lecture 3	Satisfiability	Making change
Lecture 4	Cryptography	Secret messages
Lecture 5	Byzantine generals	Liars and truth tellers
Lecture 6	Binary search	Counterfeit coin
Lecture 7	Task scheduling	Sudoku
Lecture 8	String matching	Word search
Lecture 9	Sorting networks	Rearranging trains
Lecture 10	Malfunction diagnosis	Logical reasoning

called upon to explain their answers and solution strategies. Next, somewhat more difficult or elaborate versions of the puzzle are introduced and students are again asked to solve them. As the students work on these harder puzzles, hints are revealed on the screen to help them along. This puzzles segment constitutes roughly one-third of the class duration.

In the second segment of the lecture, the instructor provides some background (origin, historical context, variations) on the puzzles and why they are deemed interesting. She or he then introduces general solution methods for the puzzles.

The third and final segment of each lecture (roughly one-third of the class duration) is spent discussing why the puzzles are relevant or similar to computer engineering problems and how solution methods for the puzzles can be adapted to these engineering and research endeavors.

Part of the first class session is devoted to introducing the course, describing its requirements, and defining the nature of puzzling problems. The discussion part of this first lecture is thus shorter than a typical session. Similarly, the tenth lecture is a bit shorter to accommodate the administration of instructor and course evaluation surveys, which is a requirement for every course taught at UCSB.

Implementation and future plans

The course described in the preceding pages has already been offered twice (Parhami, 2008) at UCSB under the title “ECE 1: Ten Puzzling Problems in Computer Engineering”. The two offerings, in spring 2007 and spring 2008, had 41 and 32 enrolled students, respectively, along with a number of students who just sat in. A formal evaluation of the course, in terms of its outcomes and curricular impacts, should be conducted after the course has been offered a few more times and the early enrollees have passed through upper-division courses in their major.

Puzzling problems in computer engineering are not limited to the 10 presented in Table 1 and the appendices of this paper. Thus, the following additional topics are under consideration. These topics may replace some of the current topics or be used to create a pool of lectures from which the instructor can pick 10 topics for any given offering of the course. This would create added flexibility for the instructor, allowing him or her to tailor the course material to the specific focal points of a particular CE programme. In the following, possible puzzles are listed in parentheses at the end of each paragraph:

- Computational geometry (Eppstein, 2008): What becomes of lines, circles, and other shapes when rendered with pixels and how to recover the original forms from their digitised versions. (Visual tricks and optical illusions)
- Loss of precision (Parhami, 2000): Seemingly simple computations that produce utterly wrong results. (Logical paradoxes and absurdities)
- Secret sharing (Shamir, 1979): How to give parts of a secret to n people so that any m of them can cooperate to discover the secret, whereas any group of $m - 1$ or fewer people cannot do so. (Anonymous complainer or whistle-blower)
- Amdahl’s law (Parhami, 2005): Why improving only one aspect of a system’s behavior may not lead to significant improvement in its overall performance. (River and bridge crossings)
- Predicting the future (Sloane, 1974): How a system can learn from past events in order to make good guesses that improve its future performance. (Determining the next term in a series)

- Circuit-value problem (Greenlaw, Hoover & Ruzzo, 1995): Example of a problem that has polynomial-time solution but that cannot be efficiently parallelised. (Parallel addition based on the carry-select principle)
- Maps and graphs (Feeman, 2002): Problems in digital representation and manipulation of maps of various kind. (Map/graph colouring and graph properties)

In the course of evaluating this paper for publication, one of the referees suggested that the puzzles used for the proposed course could have value as assignments in other courses, perhaps with a programming component. Thus, the website for our freshman seminar may be viewed as a resource for instructors of other courses. The same referee envisaged a workshop or wiki, through which ideas about suitable topics of discussion are exchanged, with each participant contributing puzzles in his/her own computing subdiscipline. In this spirit, the author would welcome suggestions from computer science and engineering instructors and from other interested parties.

Note

1. For more information on this seminar course, see its website at: http://www.ece.ucsb.edu/~parhami/ece_001.htm

References

- Aaronson, L. (2006). Sudoku science: A popular puzzle helps researchers dig into deep math. *IEEE Spectrum*, 43(2), 16–17.
- Akbulut, A.Y., & Looney, C.A. (2007). Inspiring students to pursue computing degrees. *Communications of the ACM*, 50(10), 67–71.
- Appalachian State University (2008). *Welcome to freshman seminar*. Retrieved October 31, 2008, from: <http://www.freshmanseminar.appstate.edu/firstconnections.html>
- Averback, B. (1980). *Problem solving through recreational mathematics*. San Francisco: W.H. Freeman and Company.
- Eppstein, D. (2008). *The geometry junkyard*. Retrieved October 31, 2008, from: <http://www.ics.uci.edu/~eppstein/junkyard/>
- Feeman, T.G. (2002). *Portraits of the earth: A mathematician looks at maps*. Providence, RI: American Mathematical Society.
- Franklin, S., Peat, M., & Lewis, A. (2003). Non-traditional interventions to stimulate discussion: The use of games and puzzles. *Journal of Biological Education*, 37(2), 79–84.
- Gnädig, P., Honyek, G., & Riley, K. (2001). *200 puzzling physics problems, with hints and solutions*. Cambridge, UK: Cambridge University Press.
- Greenlaw, R., Hoover, H.J., & Ruzzo, W.L. (1995). *Limits to parallel computation: P-completeness theory*. New York: Oxford University Press.
- Hayes, B. (2007). Trains of thought: Computing with locomotives and box cars takes a one-track mind. *American Scientist*, 95(2), 108–113.
- Hill, J.M.D., Ray, C.K., Blair, J.R.S., & Carver, C.A. Jr. (2003). Puzzles and games: Addressing different learning styles in teaching operating systems concepts. *ACM SIGCSE Bulletin*, 35(1), 182–186.
- IEEE-CS/ACM Joint Task Force on Computing Curricula (2002). *Computing curricula 2001: Computer science*. New York: IEEE Computer Society Press.
- IEEE-CS/ACM Joint Task Force on Computing Curricula (2006). *Computer engineering 2004: Curriculum guidelines for undergraduate degree programs in computer engineering*. New York: IEEE Computer Society Press.
- Jain, B. (2007, September 12). IT skills shortage costing Europe “billions”. *ZDNet Technology News*. Retrieved October 31, 2008, from: http://news.zdnet.com/2110-9595_22-6207460.html
- Levitin, A. (2003). *Introduction to the design & analysis of algorithms*. Reading, MA: Addison-Wesley.

- Levitin, A., & Papalaskari, M.-A. (2002). Using puzzles in teaching algorithms. *Proceedings of the ACM SIGCSE Conference on Computer Science Education*, 292–296.
- Müller-Merbach, H. (1975). The role of puzzles in teaching combinatorial programming. In B. Roy (Ed.), *Combinatorial programming: Methods and applications* (pp. 379–386). Boston, MA: D. Reidel Publishing Company.
- National Association of State Chief Information Officers (2007). *State IT workforce: Here today, gone tomorrow?* US NASCIO's National Survey of the States. Lexington, KY: NASCIO.
- O'Connor, M., & Williams, J. (2007). *Freshman seminar instructor training manual*. Temple University, Office of the Vice Provost. Philadelphia: Temple University.
- Parhami, B. (2000). *Computer arithmetic: Algorithms and hardware designs*. New York: Oxford University Press.
- Parhami, B. (2005). *Computer architecture: From microprocessors to supercomputers*. New York: Oxford University Press.
- Parhami, B. (2008). *Ten puzzling problems in computer engineering*. Retrieved October 31, 2008, from: http://www.ece.ucsb.edu/~parhami/ece_001.htm
- Parker, J. (1955). The use of puzzles in teaching mathematics. *Mathematics Teacher*, 48, 218–227.
- Princeton University (2008). *The program of freshman seminars in the residential colleges*. Retrieved October 31, 2008, from: <http://www.princeton.edu/pr/pub/fs/>
- Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11), 612–613.
- Sloane, N.J.A. (1974). Find the next term. *Journal of Recreational Mathematics*, 7(2), 146.
- University of California, Santa Barbara (2006). Computer science at the crossroads: Achievements and challenges. Retrieved October 31, 2008, from: <http://www.freshsem.ucsb.edu/seminars20052006/spring2006/int94jn.php>

Appendix 1. Easy, hard, impossible!

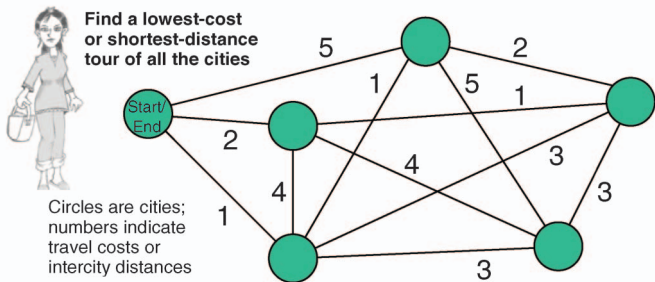
Three number sequences are specified, using very similar structures (see Figure 2). The first sequence leads to the determination of the greatest common divisor for two numbers, and participating students need a couple of minutes to realise this. The second is the Fibonacci sequence. Because this sequence does not end, the students are asked to think about how to determine the j th term or the first term that exceeds a given integer L . It is fairly unlikely that the students can come up with a method to solve this problem in class (excluding, of course, the obvious method of computing the terms, one by one, until the desired term has been reached). Once the formula for the j th Fibonacci number is revealed, the students appreciate its beauty and how it saves a great deal of time compared with the brute-force solution. The third example, Collatz's sequence (Feinstein, 2006), looks no harder than the first two. When, after some interaction in class, the students are told that no one knows whether all starting numbers lead to the same ending in this sequence, they find it difficult to believe. Near the end of the lecture, it is shown that solving Collatz's conjecture is intimately related to the programme termination (halting) problem. Presentation of two other innocent-looking but computationally hard problems, the subset sum problem and the travelling salesperson problem, concludes the lecture.

Euclid: Form a sequence of number pairs (integers) as follows:
 Begin with any two positive numbers as the first pair
 In each step, the next number pair consists of (1) the smaller of the current pair of values, and (2) their difference
 Stop when the two numbers in the pair become equal

Fibonacci: Form a sequence of numbers (integers) as follows:
 Begin with any two numbers as the first two elements
 In each step, the next number is the sum of the last two numbers already in the sequence
 Stop when you have generated the j th number (j is given)

Collatz: Form a sequence of numbers (integers) as follows:
 Begin with a given number
 To find the next number in each step, halve the current number if it is even or triple it and add 1 if it is odd
 Stop when you get to a repeating pattern such as 1, 4, 2

(a) Easy, not so easy, very hard.



(b) The travelling salesperson problem is very hard.

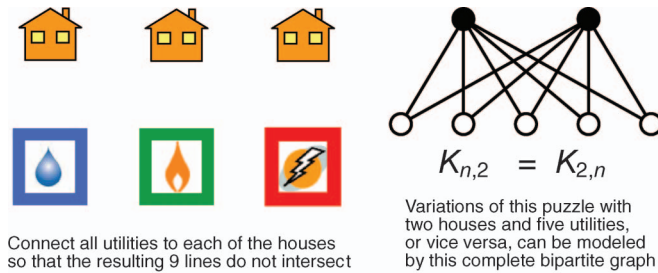
Figure 2. Problems that are equally simple to describe and comprehend can be vastly different in difficulty. Certain very hard problems are of great practical importance.

Reference

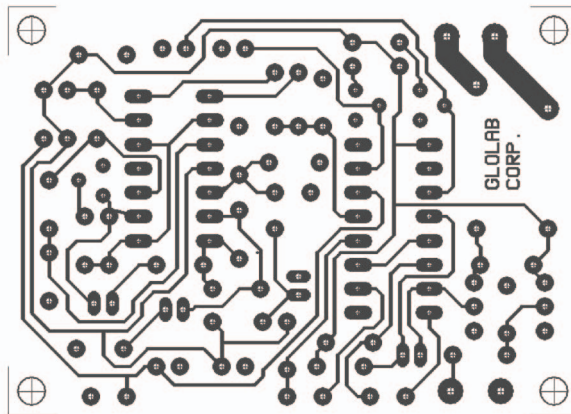
Feinstein, C. A. (2006). *The Collatz $3n+1$ conjecture is unprovable*. Retrieved October 31, 2008, from: http://arxiv.org/PS_cache/math/pdf/0312/0312309v16.pdf

Appendix 2. Placement and routing

Imagine two utility companies and five houses. Is it possible to connect each utility to every house in a manner that the 10 lines drawn do not intersect? Students quickly discover that a solution does exist, not only with five houses but with any number of houses that require connection to two utilities. In graph theoretic terms, the complete bipartite graph $K_{2,n}$ is planar (see Figure 3). What about the case of three houses and three utilities? A few minutes of experimentation convinces the students that this classic form of the puzzle has no solution, although they may not be able to provide a convincing proof; they just know that they keep getting stuck when they try to draw the last line. Presentation of a simple and elegant proof for the nonplanarity of $K_{3,3}$, using Euler’s formula for planar graphs (The Geometry Junkyard, 2005) reinforces this intuition. Nonintersecting connections are also required for circuits that must be deposited (“printed”) on a surface. If the six-vertex graph $K_{3,3}$ is nonplanar, then there is little hope that the very intricate connections of a typical electronic circuit can be drawn without intersecting each other. This leads to a discussion of multilayer printed circuits (such as a two-sided printed circuit board) and the flexibility provided by more than two layers on modern PC boards and integrated circuit chips.



(a) Two versions of the “houses and utilities” puzzle.



(b) Wires or connectors on a simple printed-circuit board.


Figure 3. Connecting utilities to houses with lines that do not cross is akin to connecting points on a circuit board with wires that do not overlap. Modern integrated circuits and circuit boards have multiple layers of wiring and are thus not limited by the non-intersection requirement.

Reference

The Geometry Junkyard. (2005). *Nineteen proofs of Euler’s formula: $V - E + F = 2$* . Retrieved October 31, 2008, from: <http://www.ics.uci.edu/~epstein/junkyard/euler/>

Appendix 3. Satisfiability

Making change for a certain sum of money using a designated number of coins can be formulated in terms of a Diophantine equation with constraints (see Figure 4a). Such equations involving integer-valued variables may or may not have solutions (be “satisfied”), but when they do, they usually admit a family of solutions. After allowing the students to work on this puzzle and find at least one solution, they are introduced to the tree-search method for finding all possible solutions. This leads into a discussion of brute-force solution strategies and why some problems do not admit more efficient solution methods (NP-complete problems, in complexity theory parlance). The special case of Diophantine equations with two-valued (0 and 1) or Boolean variables is introduced as practically important for generating tests for logic circuits and other application domains. Of course, freshmen have not yet studied logic circuits, so outputs of the AND, OR, and NOT logic elements are defined by algebraic, rather than Boolean, equations. For example, the output f of the circuit shown in Figure 4b is given by the algebraic equation $ab + (1 - b)(1 - c)(1 - d)$, where “+” represents addition rather than logical OR. In this small example, it is easy to see that the equation is satisfied for $a = b = 1$ (c and d do not matter) or $b = c = d = 0$ (a does not matter).



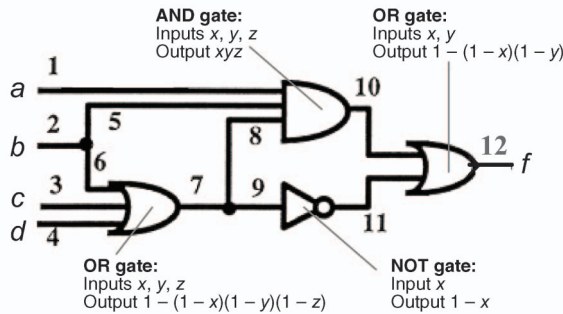
Warm-up puzzle: What is the maximum sum of coins you can have in your pocket without being able to make exact change for one dollar?

Main puzzle: Is it possible to make change for one dollar with exactly 22 coins?

Main puzzle as a satisfiability problem:
Is there a solution to $25Q + 10D + 5N + P = 100$ that satisfies the constraint $Q + D + N + P = 22$?
This is equivalent to the *Diophantine* equation $24Q + 9D + 4N = 78$ having nonnegative integer solutions

In these puzzles, “change” or “coins” means pennies, nickels, dimes, and quarters

(a) Making change provides many interesting puzzles.



Boolean satisfiability: Is there an assignment of 0 and 1 values to the circuit inputs (a, b, c , and d , in this example) that would make the circuit output 1?

The ability to efficiently solve this problem is important for circuit testing

(b) Boolean satisfiability is a key problem in many contexts.

Figure 4. Satisfiability is an important problem in many areas of computer engineering. Other than the puzzles listed here, students are introduced to an interactive online game based on satisfiability (Roussel, 2008).

Reference

Roussel, O. (2008). *The SAT game*. Retrieved October 31, 2008, from: <http://www.cril.univ-artois.fr/~roussel/satgame/satgame.php?level=1&lang=eng>

Appendix 4. Cryptography

The simplest codes for secret communication, substitution ciphers, provide a good source of engaging puzzles (see Figure 5). Students participate in decoding a few simple substitution and other ciphers, before being introduced to their weaknesses. Though decoding such ciphers may take hours by hand, or prove impossible in some cases, programmed solution methods (using a combination of exhaustive search and statistical analyses based on the frequencies of letters, letter pairs, triplets, and so on) make decoding simple and thus limit their practical value. Students are shown pictures of a few interesting cipher machines, including the ancient cylindrical device composed of rotating wheels, and the German Enigma whose very sophisticated code was broken by a team of British mathematicians led by Alan Turing (Sale, 2008). Key-based ciphers are then introduced and their encoding and decoding algorithms discussed, including those of the data encryption standard (DES). Such ciphers are less vulnerable to statistical attacks, especially if the key is fairly long. However, the need for regular change in keys (for the same reason as password changes) creates the burden of key interchange among communicating parties and possible breach of security during exchange. This has led to public-key cryptography. The lecture ends by showing how public-key cryptography works and how it enables the use of electronic signatures for authentication.

Decipher the following text, which is a quotation from a famous scientist.
Clue: Z stands for E

“CEBA YUC YXSENM PDZ
 SERSESYZ, YXZ QESOZDMZ PEJ
 XOKPE MYQGSJSYA, PEJ S’K
 EICY MQDZ PLCQY YXZ RCDKZD.”
 PBLZDY ZSEMYZSE

Cipher device: The wheels are turned until the desired message appears in a row. Letters from a different row are then used as the enciphered message

(a) Ciphers and cipher devices have been used for centuries.

	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	
	00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25	
Plaintext:	A T T A C K A T D A W N	↑ Numerical codes for the 26 letters of the English alphabet
	00 19 19 00 02 10 00 19 03 00 22 13	
Secret key:	O U R K E Y O U R K E Y	13 + 24 = 11 mod 26
	14 20 17 10 04 24 14 20 17 10 04 24	
Mod-26 sum:	14 13 10 10 06 08 14 13 20 10 00 11	11 - 24 = 13 mod 26
Ciphertext:	O N K K G I O N U K A L	
Secret key:	O U R K E Y O U R K E Y	The secret key "OURKEY", is repeated to match the message length
	14 20 17 10 04 24 14 20 17 10 04 24	
Mod-26 diff.:	00 19 19 00 02 10 00 19 03 00 22 13	
Recovered plaintext:	A T T A C K A T D A W N	

(b) Key-based ciphers are harder to crack than substitution ciphers.

Figure 5. Secret communication has been of interest since ancient times. Modern computers controlling sensitive data banks or e-commerce sites use encryption to ensure data integrity and privacy.

Reference

Sale, T. (2008). *The Enigma Cipher Machine*. Retrieved October 31, 2008, from: <http://www.codesandciphers.org.uk/enigma/index.htm>

Appendix 5. Byzantine generals

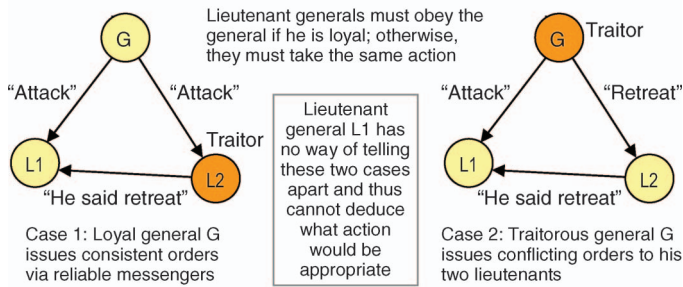
The setting for “liars and truth-tellers” puzzles (Saka, 2007) is introduced and a few easy to moderately hard puzzles are solved with student participation (see Figure 6). Then, the presence of a third tribe, the “randoms”, is postulated. Members of this new tribe lie occasionally, rather than consistently. Puzzles then show that randoms are much harder to deal with than liars. This leads naturally into a discussion of Byzantine generals camped outside an enemy city and communicating via messengers in their attempt at reaching agreement on a course of action: attack or retreat. Because some of the generals may be traitors who send inconsistent messages in their attempt to confuse the loyal generals, a protocol is needed to ensure that all loyal generals obey the commanding general if he is loyal or else they all decide on the same course of action (does not matter whether they attack or retreat in this case). A partial attack would be disastrous, because the enemy is too strong to be defeated with partial force. Near the end of the lecture, it is shown how the Byzantine generals problem represents the difficulty of reaching agreement (needed, e.g., in transaction processing) in a distributed computing environment.

Setting: You are on an island populated by two tribes. Members of one tribe consistently lie. Members of the other tribe always tell the truth. Tribe members can recognize one another, but you can't tell them apart

Warm-up: You run into a woman on the island. What single question can you ask her to determine whether she is a liar or a truth-teller?

Puzzles: (1) You meet two people A and B on the island. A says, “both of us are from the liars tribe.” Which tribe is A from? What about B?
 (2) You meet two people C and D on the island. C says, “It is not the case that both of us are from the truth-tellers tribe.” Which tribe is C from? What about D?
 (3) You meet two people E and F on the island. Each of the two makes a statement. E says, “We are from different tribes.” F says, “E is from the liars tribe.” Which tribes are E and F from?

(a) Determining who is a liar and who tells the truth.



(b) The dilemma of three generals of the Byzantine army.

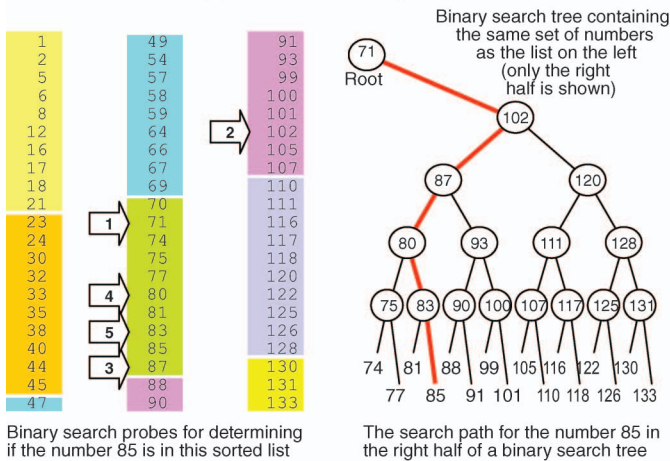
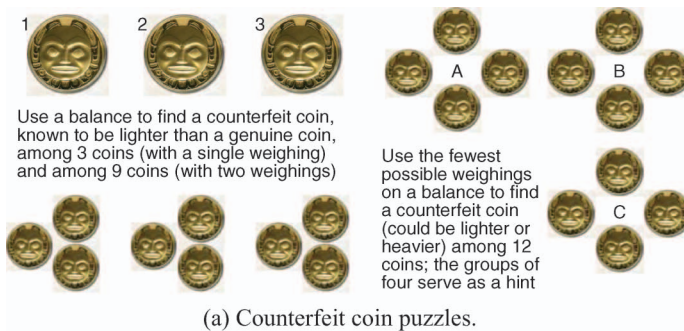
Figure 6. The Byzantine generals problem adds a twist to the “liars and truth-tellers” puzzle. Traitorous generals (representing malicious or worst-case failures in interconnected systems) lie, but not consistently, and are thus harder to deal with than consistent liars (detectable when in minority).

Reference

Saka, P. (2007). *How to think about meaning*. New York: Springer.

Appendix 6. Binary search

Imagine being presented with three gold coins, one of which is known to be a counterfeit that is lighter than the two genuine coins. How can you identify the counterfeit coin by means of a balance (that allows you to determine whether two things weigh the same) only once? More complicated versions of this puzzle postulate that the counterfeit coin is different, but not necessarily lighter. Students participate in solving several such puzzles before being shown how the solution method subdivides the search space until a single candidate remains (Du & Hwang, 2000). This leads to a discussion of binary search algorithm for finding an item within a sorted or ordered list of items (Figure 7). Examples include finding the name of a student in the class roster and looking up a word in a dictionary. It is then mentioned that this method, though quite efficient for a static list, creates a heavy overhead of maintaining the sorted order of the list when it is highly dynamic. In the extreme case when updates and searches alternate, an unordered list with linear search or an ordered list with binary search have comparable performance characteristics. Binary search trees are then introduced as a way to make both searches and updates efficient. The lecture concludes by showing how the binary search technique can be used in other contexts, such as finding a root of an equation.



(b) Binary search in a linear sorted list and in a tree.

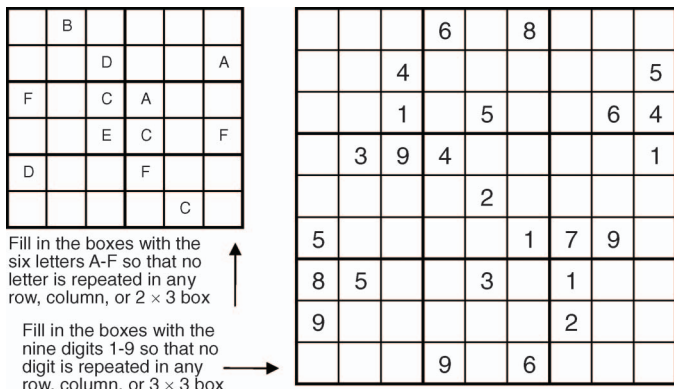
Figure 7. In solving counterfeit-coin puzzles, one uses a divide-and-conquer process to pinpoint the counterfeit coin, much as one converges to a sought element in binary search of a sorted list.

Reference

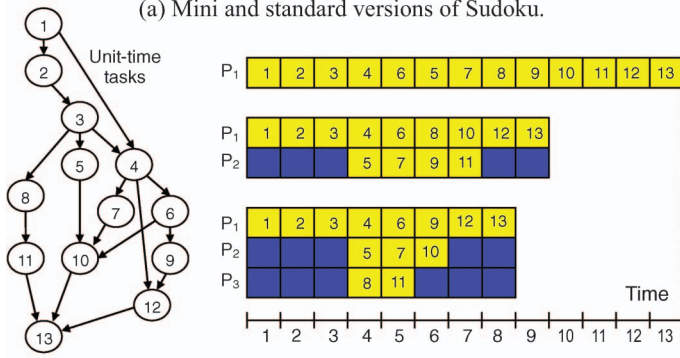
Du, D-Z. Hwang, F. K. (2000). *Combinatorial group testing and its applications* (2nd ed.). Singapore: World Scientific.

Appendix 7. Task scheduling

Mini and standard versions of Sudoku, a puzzle that has become quite popular in recent years (see Web Sudoku, a website offering free online Sudoku games in four levels of difficulty: easy, medium, hard, evil. <http://www.websudoku.com/>), are introduced (see Figure 8) and students spend a few minutes on filling the puzzle entries. Easy and difficult Sudoku puzzles are characterised and methods for solving them outlined. The problem of scheduling is introduced in terms of assigning required computer engineering courses to different quarters, while ensuring that a per-quarter unit limit is honoured and all prerequisites for each course are taken before the course. The constraints for entries in Sudoku puzzles are likened to those present in task scheduling (Aaronson, 2006). It is noted that scheduling theory predates electronic computing. Job shop scheduling, where machines in a shop must be used with great efficiency in completing tasks that require various amounts of time on different machines, produced much of the basic theory that is in use today. Scheduling of trucks and other resources entails similar techniques. Different types of tasks (unit-time or arbitrary, with or without release times and deadlines), task interdependence (reflected in a task graph), scheduling discipline (preemptive versus nonpreemptive), and criteria for judging the quality of a schedule are discussed briefly. The difficulty of optimal scheduling, and the consequent need for heuristics, is discussed and a list-scheduling scheme is presented via an example as a representative class of heuristics.



(a) Mini and standard versions of Sudoku.



(b) A task graph and its scheduling on one, two, or three processors.

Figure 8. The constraints to be satisfied when solving a Sudoku puzzle are quite similar to constraints encountered in scheduling dependent tasks on a multiprocessor system.

Reference

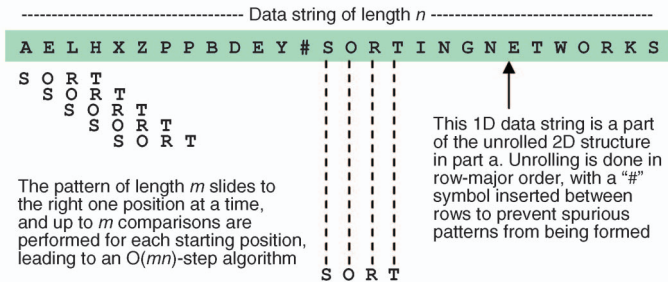
Aaronson, L. (2006). Sudoku science: A popular puzzle helps researchers dig into deep math. *IEEE Spectrum*, 43(2), 16–17.

Appendix 8. String matching

Word search puzzles are among the most familiar for students, because they are used extensively in K–12 grades as learning tools. Teachers may create (using free services on the Internet; see Discovery Education’s Puzzlemaker, a website with online tool for creating word search and other puzzles. <http://puzzlemaker.discoveryeducation.com/>) word search puzzles containing key terms and concept from a recent lesson in order to reinforce student learning. The word search puzzle shown in Figure 9 was created in this way. An interesting version of the word search puzzle does not provide a word list but rather supplies clues as to what must be located (for example, names of four countries in Europe). Such puzzles, though more interesting and challenging, are combinations of standard word search and crossword puzzles and are thus less relevant to the topic of string matching. After noting that 2D word search puzzles can be readily converted to 1D string matching problems via unrolling the 2D structure in row-major, column-major, or diagonal order, the problem of finding a pattern string within a data string is introduced. The brute-force approach (sliding window algorithm) to string matching is discussed and the need for more efficient algorithms is established. The two approaches of preprocessing the pattern to derive a finite-state machine for an $O(n)$ -step algorithm, and preprocessing the data string to generate various indices for subsequent use with different patterns are introduced. Some Google searches are performed to accentuate the practical application of indexing for speeding up searches.

<p>Word search puzzle featuring ten puzzling problems in computer engineering</p> <p>Word list:</p> <p>BINARY SEARCH BYZANTINE GENERALS CRYPTOGRAPHY EASY HARD IMPOSSIBLE MALFUNCTION DIAGNOSIS PLACEMENT AND ROUTING SATISFIABILITY SORTING NETWORKS STRING MATCHING TASK SCHEDULING</p>	<pre>V K T A S K S C H E D U L I N G J Y Y M L Y O W F H L X E Z P X F L O N T D A G L E A S Y H A R D I M P O S S I B L E Y O O F B H Q R J W H S B F N L T F T H B X M G Y L D E R S K R A C I N U D P T B E O N N P G N P S J F J B Z N O A E B Y C O I A A B E H E U K A Q C B R T M H H Q R H Z K A G T D A I H T Y G D Y C R D B S C F K U E I G F Q I M O K N R U Y T Y I T H Z S N Z S T O A T X V A C J S W X P A L S M I I S N T P Z J E T E E Q Z W W M S L K T Q D J Y U W S G N G K W J B O G W F A N I F R J W Y F Q E C X S Q Z A N C S J A J C D J R B Z U M I T N Q O Y I N X G Z Y F L A E M B X E G C Y J R W R Y N W Y A N N C E U D N C D K C L D K T O H P B B I B M V F B C A D W P G Q Q S O U C L B C V L L J L R L N Q F W C I N Z P J V E A E L H X Z P P B D E Y S O R T I N G N E T W O R K S X J I Y Q</pre>
---	--

(a) Word search puzzles involve 2D string matching.

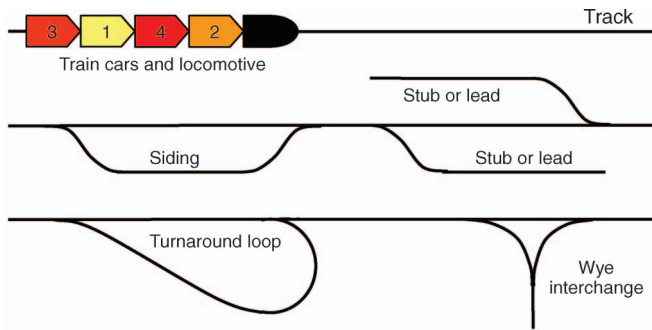


(b) Simple sliding-window algorithm for string matching.

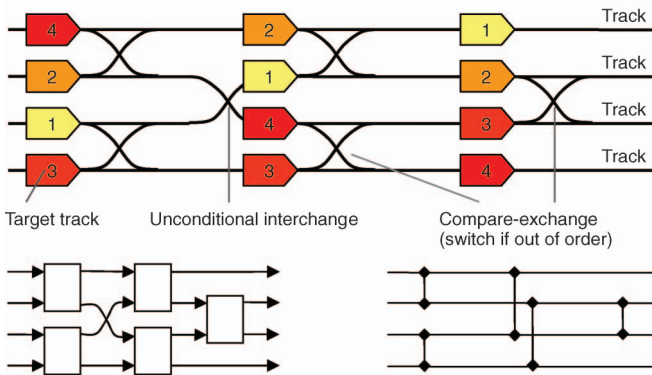
Figure 9. Finding a short string (pattern) in a much longer string (data) is like finding a needle in a haystack. The naïve sliding-window algorithm can be improved upon in many ways.

Appendix 9. Sorting networks

Sorting or rearranging train cars, using various types of attached side tracks, produces many interesting puzzles (Hayes, 2007). A few simple puzzles, such as sorting the numbered cars of Figure 10a, so that car 1 is attached to the locomotive and car 4 is at the end of the train, are presented to students and several minutes are allowed for their solution. Then, it is observed that a stub (lead) acts as a stack or last-in, first-out (LIFO) structure, while a siding is capable of acting as a queue or first-in, first-out (FIFO). Implications of these observations on the sorting algorithm are discussed. The puzzle segment of the lecture culminates in a train passing puzzle: two trains must pass each other on a single track using a siding that can hold only one car or one locomotive, and nothing else. The parallel sorting problem of Figure 10b is then introduced. Labelled train cars are on numbered parallel tracks, one car per track, and after a number of compare-exchange steps, each car must appear on the track bearing the same label. A compare-exchange step lets a pair of cars straight through if they appear in proper order and exchanges their tracks if they are out of order. This leads naturally into topic of sorting networks built of two-input sorters (comparators) as building blocks. It is noted that synthesis and verification of sorting networks is nontrivial. The 0-1 principle (Parhami, 1999) is introduced as a design and verification aid and is applied to a few simple example networks.



(a) Some facilities for rearranging train cars.



(b) Three representations of a four-input sorting network.

Figure 10. Rearranging train cars using stubs (leads), sidings, turnaround loops, and Wye interchanges leads into a discussion of sorting algorithms and sorting networks.

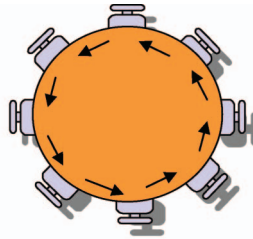
References

Hayes, B. (2007). Trains of thought: Computing with locomotives and box cars takes a one-track mind. *American Scientist*, 95(2), 108–113.
 Parhami, B. *Introduction to parallel processing: Algorithms and architectures*. New York: Plenum Press.

Appendix 10. Malfunction diagnosis

Logical reasoning is used to derive information about the health of a distributed computing system from local testing outcomes. This topic is introduced via a puzzle in which a group of people sharing a common skill or attribute try to determine if there are any impostors among them. For example, medical doctors (MDs) may ask specialised questions of a person in order to determine whether she or he is a legitimate MD or an impostor. Suppose that the diagnostic power of such a question is 100%, meaning that an MD can determine for sure whether another person is an MD. In the simplest case, each person asks one question of a single person, as depicted in the circular arrangement of Figure 11. The students readily determine that a single impostor is always correctly identified by analysing the outcomes of the n questions, whereas two or more impostors would create a problem. After posing a number of other puzzles of this type, the instructor relates the puzzle to malfunction diagnosis in distributed multicomputer systems (Somani, Agarwal & Avis, 1987) where each machine is capable of testing a number of other machines and the diagnostic syndrome (a string of 0s and 1s representing the test outcomes) is used to identify the malfunctioning unit(s) via deduction or by consulting a syndrome dictionary. Deriving malfunction diagnostic capabilities from a testing graph, and the inverse problem of designing a suitable testing graph for a desired diagnostic capability, are then discussed.

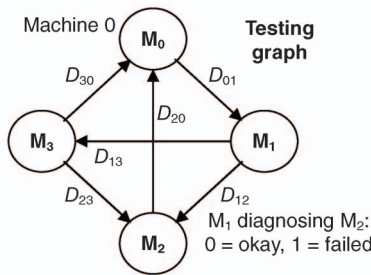
At a round dinner table, n people of a certain profession (say, computer engineers) try to determine if there are any impostors among them. Each asks the person to his or her right a question and renders a judgment.



Assumption: A genuine person always arrives at the correct judgment about another one, but an impostor's judgment is erratic.

What is the maximum number of impostors that can be correctly identified?

(a) Identifying impostors at the dinner table.



Syndrome dictionary:						
D_{01}	D_{12}	D_{13}	D_{20}	D_{30}	D_{32}	Malfn.
0	0	0	0	0	0	---
0	0	0	1	1	0	M_0
0	0	1	0	0	0	M_3
0	0	1	0	0	1	M_3
0	0	1	0	1	0	M_3
0	0	1	0	1	1	M_3
0	1	0	0	0	1	M_2
0	1	0	1	0	1	M_2
1	0	0	0	0	0	M_1
1	0	0	1	1	0	M_0
1	0	1	0	0	0	M_1
1	1	0	0	0	0	M_1
1	1	1	0	0	0	M_1

Practical malfunction diagnosis has two sub-problems: (1) Given a testing graph, which failure patterns are diagnosable? (2) For a particular diagnostic capability, what testing relationships are needed?

(b) Malfunction diagnosis in a multicomputer system.

Figure 11. Malfunction diagnosis requires logical reasoning based on test outcomes. The problem is akin to identifying all the randoms among a group of truth-tellers and randoms (see Appendix 5).

Reference

Somani, A. K., Agarwal, V. K. & Avis, D. (1987). A generalized theory for system level diagnosis. *IEEE Transactions on Computers*, 36(5), 538–546.