The Specker Challenge Game for Education and Innovation in Constructive Domains

Karl Lieberherr and Ahmed Abdelmeged and Bryan Chadwick

College of Computer & Information Science
Northeastern University, 360 Huntington Avenue
Boston, Massachusetts 02115 USA.
{lieber,mohsen,chadwick}@ccs.neu.edu

October 5, 2010

1 Introduction

We have the following vision: To teach a constructive domain (e.g., domain in computer science, mathematics, engineering, etc.), we design a domain-specific game where the winning students demonstrate superior domain knowledge. The game is simple: the students make constructive claims about the domain we want to teach and refute each others' claims. The students who are the most successful in defending their claims and in refuting the claims of others win the game. Some benefits of the game are: (1) the students give each other constructive feedback about their claims. Students who lose points gain knowledge to improve their game in the future. (2) Although the students operate egoistically, maximizing their points, they create a common good: knowledge. The claims that have not been successfully opposed are candidates for truth. (3) The game is fun and adjusts to the skill levels of the students. It can be played productively by undergraduates as well as by world-class researchers to push the state-of-the-art in a domain. The game also serves as a useful, and novel software development process to develop software for optimization problems.

2 Claims defined by parameterized reputation protocol

Let's assume we have two students, Alice and Bob who argue about a claim. Alice thinks the claim is right and Bob questions it. The claim is defined by an instantiation of a generic refutation protocol:

Refutation protocol (in a nutshell) Alice produces message x in X consisting of $(public(x), secret(x))$. The secret part may be empty. She sends $public(x)$ to Bob. Bob checks $public(x)$ and produces message $y(public(x))$ in $Y(X)$ and sends it to Alice. Alice checks $y(public(x))$ and reveals $secret(x)$ to Bob who checks it.
$refute(x, y(public(x)))$ implies Bob wins and else Alice wins.

The refutation protocol may be iterative, consisting of several moves. In each move, a piece of x, x_1, and a piece of $y, y_1(x_1)$ is produced.

Note that in some refutation protocols, Bob produces x and Alice produces y.

The refutation protocol is parameterized by sets $X, Y, Public(X)$ (restricting $public(x)$) and $Secret(X)$ (restricting $secret(x)$)and the $refute$ function. A claim is a tuple $(X, Y, Public(X), Secret(X), refute)$ used to instantiate the refutation protocol with an additional confidence factor in $[0, 1]$ (see below).

Alice claims that she can create an x in X so that whatever y in Y Bob creates, $refute(x, y(public(x)))$ will be false. Before Bob challenges Alice on her claim, he will carefully study the refutation protocol defining the claim to decide whether it is worth his time to challenge Alice. Only if he sees a winning strategy for him he will try to refute the claim. Alice and Bob have a clearly defined protocol to settle their argument.

Claim design must follow a few principles. Mathematical claims that come from interesting theorems with an even number of alternating quantifiers are good candidates for claims. This is folklore knowledge. We distinguish between mathematical claims and non-mathematical claims.

- Mathematical. For those claims we can determine in principle in advance that we have a winning refutation strategy or that we have a winning defense.

– Non-Mathematical. For those claims we cannot guarantee logically that we will win. We must play out the game and the refutation protocol will decide. An example of a non-mathematical claim is the claim for the Maximum Independent Set problem shown below or the claims in the renaissance game.

The claims must have the property that it is "challenging" for Alice to produce her x and for Bob to produce his y (knowing x). It must be "challenging" for Alice to produce claims that cannot be refuted. It must be "challenging" for Bob to analyze a claim to reach the refute decision. Another issue in claim design is privacy. The information that is exchanged between the players should not reveal too many secrets. Otherwise, the players are less motivated to develop new ideas.

Claim design for educational purposes may have an indirect flavor. For some simple claims, a significant amount of material needs to be mastered that is not at all mentioned in the claim formulation.

3 Game benefits

The game engages the students in a well-defined dialog about the domain. We view this as a useful property for online education where student-to-student interaction is important. Although a game results in a winner and loser (ties may also happen but are ignored in this introductory discussion), the loser gains valuable information from the winner. If Alice loses, she can study the y she got from Bob that made her lose. Maybe the y has a structure that gives a hint at Bob's clever technique for producing y. If Bob loses, he can study the x that Alice created.

Consider claims of the following algorithmic form: Alice claims she has an algorithm that solves any problem x in X with a solution $y(x)$ in $Y(X)$ that has quality $q(X)$. The refutation protocol is: Bob constructs a hard problem instance x_0 in X and Alice must find a solution y of the claimed quality. If she cannot, Bob wins. Games of this kind drive the scholars towards a common good, although they behave egoistically: to find the best algorithm and to pose the hardest problems. The scholars create social welfare through their egoistic behavior thanks to the mechanism design the game uses.

It is important to notice that the game does not force the students to proving or disproving their mathematical claims (unless the protocol explicitly exchanges proofs). They only need to be capable of defending their claim if another student tries to refute it. Of course, the defending/refuting activity invites the students to think about proving their claims. If a claim is defended (Alice wins), it does not mean it is true. It could be that Bob made a mistake. If a claim is refuted (Bob wins), it does not mean it is false. It could be that Alice made a mistake. However, we can state the following properties of the game: If Alice is perfect and Bob wins, the claim is false. If Bob is perfect and Alice wins, the claim is true. By "perfect" we mean making the best decisions to win the game.

4 Examples

4.1 O Notation

How can we engage our students in a constructive discussion about the big O notation. The instructor prepares for them a set of claims to choose from and Alice chooses the following claim: $n^{1/2} = O((log(n))^{10})$. The parameters to the refutation protocol are: $X = \mathbb{R}^+ \otimes \mathbb{R}^+$, where $\mathbb{R}^+$ is the set of positive real numbers, and $Y = \mathbb{R}^+$. There are no secrets in this case. The refute function is $refute((C, n_0), n) = n \geq n_0 \wedge n^{1/2} > C \cdot ((log(n))^{10})$. The refutation protocol is: Alice finds $x = (C, n_0)$ and Bob finds $n > n_0$ so that refute returns true. After playing this game a few times, Bob starts to win consistently and they start looking for a reason. They remember a theorem ...

4.2 Highest Safe Rung Problem

How can we engage our students in a constructive discussion about search, from linear search to binary search and what is between? Along the way, we want to teach them about binary decision trees and attempting to minimize the longest root-to-leaf path in such trees. More importantly, they learn about deriving and solving equations and analyzing the asymptotic behavior of functions. We do this by engaging the students in the following game between Alice and Bob. The game is based on the following problem from [2]: You are doing some stress-testing on various models of

glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with n rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the highest safe rung. You have a fixed "budget" of $k \geq 1$ jars. By $HSR(n,k) = q$ we summarize Alice' claim that she can determine the highest safe rung for a ladder on n rungs and a budget of k jars with at most q experiments. The parameters to the refutation protocol are: X is the set of sequences of questions to Bob of the form $breaks(j)$: Does the jar break at rung j? Y is the set of sequences of answers from Bob of the form true or false. The answers must be consistent: if $breaks(j_1)$ is true then $breaks(j_2)$ holds for all $j_2 > j_1$ and if $breaks(j_1)$ is false then $breaks(j_2)$ does not hold for all $j_2 < j_1$. The refutation function for $HSR(n,k) = q$ is $refute(n,k,q,x,y)$ returns true if (y is not consistent with x) or (y contains more than k true answers) or (x contains more than q questions) or the highest safe rung is not uniquely determined.

We give the following claims to choose from: $HSR(9,2) = 6, HSR(9,2) = 5, HSR(9,2) = 4, HSR(9,2) = 3$. Alice chooses claim $HSR(9,2) = 4$ after finding a decision tree whose longest root to leaf path is 4. Bob also tries to find a decision tree of depth 4 but he cannot. Therefore Bob decides to refute Alice' claim. However, Alice successfully

4.3 Maximum Independent Set

An independent set in a graph is a set of mutually nonadjacent vertices. The problem of finding a maximum independent set in a graph, IndSet, is one of most fundamental combinatorial NP-hard problems. The refutation protocol is: Alice constructs a graph G with at most n nodes and deposits her secret independent set I_1. Alice gives G as well as the size of I_1 to Bob. In other words, I_1 is a secret for now. Bob has 10 minutes to construct his independent set I_2 which he gives to Alice and she checks it. Alice reveals her secret set I_1 and Bob checks it. Bob wins iff $size(I2) \geq size(I1) * 0.9$. A few of the topics that students learn while playing the game: The difference between checking and searching algorithms. How to hide secrets. How to search for secrets. Approximation Algorithms.

4.4 Min-Max

Calculus is a constructive domain that can be covered with the game using modern tools like WolframAlpha. We choose claims of the form: $\forall x \in X \exists y \in Y : f(x,y) \geq c$. The claim is of the form $\forall\exists$, therefore the refutation protocol is (Alice claims, Bob tries to refute): Bob chooses x_0. Alice chooses y_0. Bob wins iff $f(x_0, y_0) < c$.

Example: Claim $GoldenRatio(c)$: $min_{x\in[0,1]} max_{y\in[0,1]}(x * y + (1-x)(1-y^2)) \geq c$, where c in $[0,1]$. If the players are perfect, claim $GoldenRatio(0.62)$ is refutable while claim $GoldenRatio(0.61)$ is not. To find out why, calculus knowledge is needed.

For mathematical claims like this, it makes sense to close them under negation and also allow:
Claim $NegGoldenRatio(c)$: $\exists x \in X \forall y \in Y : f(x,y) < c$. When Bob refutes claim $GoldenRatio(c)$ successfully, he claims a new claim $NegGoldenRatio(c)$ to find out whether his refutation was accidental.

4.5 Renaissance Games

Consider yourself moved back a few hundred years to the time of the Renaissance in Italy. The mathematicians challenged each other regarding the best algorithms to solve problems. Tartaglia played the role of Alice and Fior the role of Bob in a game about cubic equations. Tartaglia won because he had discovered a technique to solve Fior's equations while Fior could not solve the ones posed by Tartaglia.

They used the following adapted refutation protocol: x and y were delivered simultaneously, i.e., y does not depend on x. Alice solves Bob's equations and Bob solves Alice' equations. $X = Y$ = sets of cubic equations. x = thirty cubic equations from Alice. y = thirty cubic equations from Bob. $refute(x,y)$ = Alice solves fewer equations in y from Bob than Bob solves in x from Alice within two hours.

Note that those renaissance claims are not mathematical claims in the sense of the Highest Safe Rung claims or the Min-Max claims. Tartaglia could not be a 100% sure that he would win. But he thought that Fior had not discovered the solution technique that he had and this gave Tartaglia a significant edge. But the game is interesting and the winner demonstrates that he/she has better algorithms.

5 Game Versions

The above game family describes the core of the Specker Challenge Game.

5.1 Specker Challenge Game Intensional

For the full version, we use the core game to create a world of scholars who propose and oppose claims. There is a language of claims defined by the game designer that permits a huge number of claims intensionally, rather than extensionally.

5.2 Specker Challenge Game Extensional

For some applications of the Specker Challenge Game, it is better to formulate a heterogeneous market of claims from which the scholars must take the claims they propose. Instead of defining the allowed claims intensionally through a predicate, they are defined extensionally by enumerating them. The market of claims is defined by the game designer.

SCG is also an acronym for Scientific Community Game. Indeed the scholars compete for reputation and they build, if the claims are mathematical, a knowledge base of unopposed claims.

5.3 SCG/Board

This is the informal version of the game where two students challenge each other and together check the rules of the game. SCG/BoardExtensional is a convenient combination because of its low overhead. The students can control their educational experience by choosing claims in a domain they want to explore further.

5.4 SCG/Avatar, a Web Application

The main attraction of SCG/Avatar is that players get the opportunity to create their "organism" that needs to fend for itself in a real virtual world inhabited by organisms created by other players.

T his version of the game works well when the domain is sufficiently simple so that the task of proposing and opposing can be easily programmed. SCG/Avatar brings to the game a strong software development component and a very fun competitive/collaborative environment. The avatars live somewhere on the web, written in your programming language of choice, and compete with each other in a full round-robin or Swiss style tournament. We have observed that students during the running game changed their avatar to improve its performance against the avatars of their peers. When they come to class, they get a simple baby avatar that can walk and talk (it can briefly survive on the web) but does not have much intelligence. Then they add intelligence that is hopefully better than the intelligence that their peers add. The game creates a ranking of the avatars that gives feedback to the students about their algorithm and software development skills.

The Highest Safe Rung and Min-Max games could be easily played using SCG/Avatar by teaching the avatars enough calculus so that they propose strong claims and successfully refute weak claims. We used a more complex version version of SCG/Avatar based on the Constraint Satisfaction Domain.

5.5 Optimization Problems-Based

A large class of interesting games are obtained from optimization problems. Alice claims that she has an algorithm to solve a family of optimization problems with a given resource bound and a given solution quality. The refutation protocol: Bob gives Alice a problem (x) in the family and if Alice cannot produce a solution (y) within the given resource bound of the promised quality, Alice loses.

– NPO
How can we drive innovation towards better algorithms for NP optimization problems? NPO is the class of all NP optimization problems. The ingredients to an optimization problem are the set of input instances or problems, the set of feasible solutions, and the measure defined for a feasible solution. In a nutshell, NPO problems are optimization problems whose feasible solutions are short and easy to recognize.
Using SCG/Avatar , we design a virtual scientific community that through the avatar game improves its knowledge. The game is designed in such a way that the winning avatars have the best approximation algorithms for the NPO problems under discussion.

SCG/Avatar offers a new way to evaluate algorithms without resorting to an asymptotic analysis. Good asymptotic behavior is important but games are won also by clever implementations that use small constant factors. The problem instances allowed to appear in the game are all bounded by a large constant.

6 Axioms for Game Mechanism

The scholars must stay active. They must propose and oppose at least one claim in each round. An activity sequence of scholar Alice is described by the regular expression

$$AliceActivity = propose\ OPPOSE.$$

$$OPPOSE = (STRENGTHEN|(REFUTE\ provide\ PROVIDE))$$

$$STRENGTHEN = PROPOSE.$$

Activities in only capital letters are done by her friend Bob.

Strengthening a claim is a sort of opposing an existing claim and proposing a new (stronger) claim at the same time. That's why strengthening qualifies both as a proposition and as an opposition at the same time.

A scholar is active during a round if it engages in: (propose and refute) or strengthen.

Confidence factors are real numbers in [0,1] that indicate how confident the proposer is in the claim. The confidence factor should reflect the amount of effort that has been put into the claim. For example, one could use confidence 1 for a claim for which the scholar has a proof. Confidence 1 means that we believe that the claim cannot be opposed. The scholars are encouraged to give their "true" confidence in a claim. If not, they will be penalized.

The following axioms must be followed by any game mechanism. They are about how reputation is gained.

1. Scholars gain reputation either by opposing other scholars' claims or by having their claims recognized by other scholars (when the other scholars fail to oppose). Scholar's gain from their claim is proportional to both the credibility of their claim and the recognition of the claim after the refuting protocol. Note: the payoff for strengthening comes after the strengthened claim is recognized (successfully defended or never opposed).
2. One scholar's reputation gain is another scholar 's reputation loss. The sum of all scholars's reputation is preserved.
3. The discounting protocol recognizes claims by a recognition factor in $[-1, 1]$. A recognition factor of 1 means that the other scholar has completely failed to refute that claim. A recognition factor of -1 means that the other scholar has completely succeeded to discount that claim.
4. Claims have a credibility in $[0..]$. The Credibility of a claim made by a scholar is proportional to either the scholar's "confidence" in the claim or the scholar's reputation or both.
The claim's confidence reflects the amount of effort made by the scholar behind the claim to prove the claim (i.e., turning it into a theorem or finding other supporting evidence). Scholar's reputation is the sum of the scholar's initial reputation plus the reputation gains and losses; thus reflecting the past performance of the scholar.

Those axioms define a family of mechanisms that can be used to implement the game.

There is an issue with the axioms as they are currently formulated: When a scholar with low reputation refutes the claim of a scholar with high reputation, the low reputation scholar will significantly increase its reputation. But when the low reputation scholar loses, it will decrease the reputation by the same amount. This may not be fair.

7 Applications

7.1 Education

The Specker Challenge Game works very well to create a productive interaction between students of constructive domains, like STEM fields. The Specker Challenge Game focuses the student interaction on a specific domain and gives the students valuable hands-on experience when they propose and oppose.

- Software Development We have used SCG/Avatar successfully in software development courses. Implementations of $SCG(CSP)$ have been used at Northeastern University to teach several semesters of a senior level course on software development [4, 3]. Throughout the course students learn not only about the rigors of software organization and evolution, but about some details of SAT/CSP algorithms. The feedback from both course staff and regular competitions keeps students involved, putting their best ideas into the avatars.
  The course invites students to master a wide variety of CS topics ranging from Theory of Computation, Discrete Structures (combinations, permutations, simple combinatorics), Calculus (maximizing and minimizing multivariable rational functions) to Object-Oriented design and Software Engineering.
  An important part of teaching is to evaluate whether the students understand the computational processes and whether they can effectively use them in practice. A round-robin contest is the perfect tool to give the students quick feedback how well they do compared to the other students. Furthermore, the agents help each other with testing.
  When students come to the course, they get a baby agent that must be equipped with intelligence to win. We got reports in Spring 2009 that students were so immersed into the game that they ignored other courses and put all their effort into winning. We used the rule that after two weeks of contests, the source code had to be revealed. This on one hand lead to rapid improvements to the agents but it also contributed to the competitive nature of the game.
  We plan to further develop our use of the Specker Challenge Game for teaching Software Development. There is significant value in giving the students a baby agent that they have to nurture into an "intelligent adult" using state-of-the-art software development technology.
- Algorithms We have also used Specker Challenge Game successfully in two undergraduate algorithms courses [6]. Here we used the "board" version of the game (SCG/Board) where the two participating students jointly check the game rules. We use the game to help the students to create ideas about how to solve textbook problems. We see value in having students defend claims that are true and refute claims that are false.

7.2 Innovation

We believe that the small innovation steps we have observed in undergraduate competitions can be achieved on a larger scale when world-class researchers participate (e.g., PhD students in algorithms) both for SCG/Avatar and SCG/Board .

7.3 Hiring

Companies can benefit from SCG/Avatar by screening potential employees with the game. When a company seeks algorithmic/implementation skills in a given algorithmic problem solving domain, they define a game for this domain and let potential employees participate. The potential employees might create useful knowledge for which they are compensated. For example, the potential employee competition might create an algorithm that beats the algorithm that was created in-house.

7.4 Better Software Development Process for Computational Problems

Developing software for computational problems is an important ongoing activity. The claims are of the form: $Claim(X, Q, R)$: I have a program that solves instances in X with quality Q and resource bound R. SCG/Avatar provides a new software development process for such software that is based on both collaboration and competition. For example, instead of

having 4 people working as one team, we use 2 people to develop Alice and 2 to develop Bob and the competitions between Alice and Bob will improve them. The process works better when more avatars are involved.

The benefits of the process are:

- Engages Software Developers. Nobody wants to see their avatar suffer and being beaten by their peers. The competitions typically run over several hours and we saw software developers stay up at night and improve their avatar between rounds in the same competition.
- Meaningful Measurement. The tournament ranking provides an objective evaluation of the avatars and the teams behind them with respect to software development and algorithm skills. The game is designed in such a way that the winning avatar must demonstrate better optimization results.
- Encourages Good Software Engineering Properties. There is a lot of pressure between the competitions to update the avatar's code quickly. This can only be done if the code has good software engineering properties.
- Encourages Good Algorithms. Winning avatars often have superior algorithmic knowledge that they exploit in the game.
- Testing. The avatars are thoroughly tested through the opposing activity.

8 Evaluation of the Specker Challenge Game / Disadvantages

The Specker Challenge Game, because its very flexible refutation protocol, covers a broad area. Because of this coverage, the effort of learning the game can be quickly amortized. But one disadvantage of the game is that students and software developers need to learn the game mechanics of a scientific community.

The Avatar version requires a very well functioning game engine to work well. The administrator needs to painstakingly check all avatars whether they follow the rules. The Avatar version requires a software tool to quickly generate baby avatars. We hope to make this parameterized engine freely available but we are not there yet.

We found SCG/Avatar to be addictive. Students identified with their avatars and did not want to see them suffer. We had to advise the students to balance their efforts with other courses.

9 Origins of the Specker Challenge Game

Initially, we used the domain of Boolean CSP: Boolean constraint satisfaction problems, motivated by our joint work with Ernst Specker [8]. Specifically, our problem domain X was the set of CSP *formulas*, where a formula is a sequence of constraints, each over a set of variables. For a given CSP formula $x \in X$ we use $V(x)$ to represent the set of all variables used in its constraints. Feasible solutions, $S(x)$, are assignments to a formula's variables, defined as all maps whose domain is $V(x)$, and range is boolean values, $\{true, false\}$.

For CSP, the easiest way to define a sub-domain for a claim (*i.e.*, X') is to do so intensionally by giving a set of *relations* that can be used to create legal constraints. The function $fsat(x, s)$, which calculates the fraction of satisfied constraints in x using assignment s, is used for predicate descriptions.

A scholar might offer a claim using the relation 1-IN-3, which is *true* when exactly one of its three variables is assigned *true*. The claim proposed could be: Alice claims: There exists a CSP formula using only the 1-IN-3 relation so that for all assignments to the formula at most the fraction 0.4 will be satisfied.

$$\langle\langle\{\text{1-IN-3}\},\, Q\rangle,\, 0.4\rangle \quad \text{with} \quad Q(x,\, s) = (fsat(x,\, s) \leq 0.4)$$

If refuted by Bob, the offering scholar Alice would need to provide a CSP instance using the 1-IN-3 relation, *e.g.*:

$$x \;\equiv\; \langle\, (\text{1-IN-3}\ \{a,\, b,\, c\}),\ (\text{1-IN-3}\ \{a,\, b,\, d\}),\ (\text{1-IN-3}\ \{b,\, c,\, d\})\,\rangle$$

We can quickly see that with the defined predicate, Q, it is rather easy to refute the claim given this problem instance. Though not all assignments provide a good fraction of satisfied constraints (*e.g.*, the all *false* assignment does not satisfy any), there are multiple assignments that do. Two possible solutions and their satisfied fractions are:

$$\begin{aligned} fsat(x,\, \{\, a \mapsto false,\, b \mapsto true,\, c \mapsto false,\, d \mapsto false\,\}) &= 1 \quad \text{and} \\ fsat(x,\, \{\, a \mapsto true,\, b \mapsto false,\, c \mapsto false,\, d \mapsto false\,\}) &= 2/3 \end{aligned}$$

For formulas made only of this relation, 1-IN-3, there always exists an assignment that will satisfy at least $4/9$ of all constraints. This is known as a *P-optimal* threshold for this specific CSP problem [7]: we can satisfy this fraction quickly (in polynomial time), but to do better than $4/9 + \epsilon$ for $\epsilon > 0$ makes the problem NP-hard. We get this value by maximizing the polynomial: $3 \cdot p \cdot (1-p)^2$, which we derive from the truth table of 1-IN-3, eventually obtaining the maximum value at $p = 1/3$.

10 Related Work

TopCoder [10] has connections to SCG/Avatar. TopCoder offers algorithm competitions where you need to defend your own program and try to find bugs in programs of others. But there are big differences: Specker Challenge Game has claims which can be flexibly defined while in TopCoder they have a fixed form: I have an algorithm that solves the problem correctly. If you want to refute my claim you must provide a test-input where the code fails. Also in SCG/Avatar the avatars must create claims which is absent in TopCoder.

The Specker Challenge Game is patterned after a real scientific community: This related work is quite old. Scientists are encouraged to offer results that are not easily improved. They must offer results that they can successfully support. They strengthen results, if possible. They must stay active and publish new results or oppose current results. They want to become famous!

How to turn predicate calculus statements with an even number of alternating quantifiers into a game is well-known folklore.

[5] gives further information on the Specker Challenge Game.

[9] provides a good introduction to game design.

[1] promotes the use of serious games in computer science programs but from a different angle than this paper.

Acknowledgements We would like to thank Novartis/NIBR for partially supporting the development of the Specker Challenge Game. Alex Dubreuil was the teaching assistant for several courses where the Specker Challenge Game was used and he gave us valuable feedback and encouragement.

References

1. K. Becker and J. R. Parker. "Serious Games + Computer Science = Serious CS". *J. Comput. Small Coll.*, 23:40–46, December 2007.
2. J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
3. K. Lieberherr. Software Development: CSU 670 Fall 2009. Website, 2009. `http://www.ccs.neu.edu/home/lieber/courses/cs4500/f09/cs4500-f09.html`.
4. K. Lieberherr. Software Development: CSU 670 Spring 2009. Website, 2009. `http://www.ccs.neu.edu/~lieber/courses/csu670/sp09/csu670-sp09.html`.
5. K. Lieberherr. The Specker Challenge Game. Website, 2009. `http://www.ccs.neu.edu/~lieber/evergreen/specker/scg-home.html`.
6. K. Lieberherr. Algorithms and Data CS 4800, Fall 2010. Website, 2010. `http://www.ccs.neu.edu/home/lieber/courses/algorithms/cs4800/f10/course-description.html`.
7. K. J. Lieberherr. Algorithmic extremal problems in combinatorial optimization. *Journal of Algorithms*, 3(3):225–244, 1982. `http://www.ccs.neu.edu/~lieber/p-optimal/karl-algo-extremal.pdf` .
8. K. J. Lieberherr and E. Specker. Complexity of partial satisfaction. *Journal of the ACM*, 28(2):411–421, 1981. `http://www.ccs.neu.edu/~lieber/p-optimal/JACM1981.pdf` .
9. J. Schell. *The Art of Game Design: A Book of Lenses; electronic version*. Elsevier, Burlington, MA, 2008.
10. TopCoder. The TopCoder Community. Website, 2009. `http://www.topcoder.com/`.