Project Description

1 Introduction

Software is an increasingly important part of today's technology driven world. Before software is bought or used, evaluation of its correctness, robustness, and performance must take place. Thorough software evaluation is expensive, which is especially an issue for those dealing with computationally expensive optimization problem domains — a very important class of the more interesting and difficult computational problems. In the past, benchmarks and test suites have been developed to help with the evaluation and comparison of algorithmic software solutions to computational problems. Testing evaluates the completeness software ensuring an application is robust, while traditional benchmarks set standards for quantitative comparison and evaluation. In both cases, the difficulties of development are compounded by the fact that results are usually inflexible and unable to adapt along side an evolving application.

With this proposal we wish to address the difficulties of software evaluation by utilizing a dynamic strategy based on *artificial markets* of computational challenges. Challenges represent an agent's (and team's) constructive beliefs about their algorithms' abilities to create and solve difficult problems. Within the market, *agents* offer and accept challenges in the given problem domain. Once accepted, a specific problem matching the challenge is provided by the offering agent, then solved by the accepting agent. In turn, successful agents are rewarded for both providing difficult problems and solving other agents' problems effectively. The market itself becomes a dynamic benchmark for a particular problem domain by ranking agents based on their algorithmic success, and the problems generated within the market become a dynamic test/benchmark suite. The most difficult problems to solve within the market lead agents toward a deeper understanding of the domain and its algorithmic and computational properties.

1.1 Thesis

It is our thesis that (1) artificial markets are a useful tool to thoroughly and automatically (quantitatively) evaluate software; (2) artificial markets drive innovation, helping to find better algorithms to solve computational problems; and (3) computational artificial markets provide a platform for the sharing (and teaching) of ideas. The artificial market invites agents to expose weaknesses in other's algorithms, while teams improve their agents' solutions. The automatic running of regular *competitions* (or market contests) gives the agents frequent feedback on the quality of their algorithms, allows improvements to be tested often, and provides a meaningful measure of an algorithm's success in the problem domain. The goal of this project is to explore the validity of these claims.

1.2 Computational Artificial Markets

For this proposal we focus on artificial markets of computational challenges: agents offer, accept, and solve challenges in a given problem domain involving various algorithms. In its simplest form, a *challenge* is a pair consisting of a set of problems from the domain and a *price*. The problem set may be defined extensionally, by giving explicit instances from the domain, or intensionally, by abstractly describing a subset. When a challenge is *accepted* the price is paid, and an instance of the problem is *provided*. The accepting agent must *solve* the given problem instance, and is subsequently reimbursed based on the *quality* of the solution.

Clearly, an intelligent agent will only accept challenges for which it expects to recuperate the original price with some profit. In order to make a profit, agents must be able to solve many problems within the domain well. On the other side of the market, agents must also be able to offer challenges and provide difficult problems from the domain. Agents in both roles must be able to estimate prices correctly. The market is made interesting by the fact that the acceptor of a challenge cannot know which of a potentially large number of problems will be provided, so he must analyze the challenge definition and choose wisely.

Throughout the course of a competition (or market contest), agents get several chances to offer and accept challenges, and provide and solve problems. Once a competition is complete, agents can be ranked based on their profit (or loss), giving a quantitative evaluation of the success of their algorithms. Together, the agents' offering and providing components correspond to dynamic benchmarks, finding and creating the most difficult problems to be solved. The solution components of the agents embody the best of their algorithmic knowledge of the problem domain, a sub-domain, or even specific niches.

1.3 An Alternative View

The contributions of this proposal fall in the space of creating better benchmarks for algorithmic evaluations, driving both innovation and analysis. The artificial market ranks agents based on their ability to offer and accept computational challenges, and provide and solve related problem instances. As an alternative view of this proposal, we can see the market as a new kind of *game* and study *mechanism design* for it, sometimes called *inverse game theory* [10]. We agree with Papadimitriou's statement, "*All design problems are now mechanism design problems*", but we believe that algorithm design problems— finding effective algorithms irrespective of their widespread use— is a mechanism design problem. The task is to design a game, given desired goals, in such a way that individual players achieve the goals motivated solely by self-interest.

We propose to study the design of families of related mechanisms, or *meta mechanism design*. In traditional mechanism design, one designs a mechanism for a specific problem; like distributing a shared good. An artificial market can be seen as a meta mechanism design: given a problem domain description we can create a game that will help us find better algorithms for the domain. One reason why the artificial market succeeds is because it creates situations where both winning and losing agents gain information, and the market as a whole benefits from the information (profit and loss) of all agents. There are 2 interesting cases:

1. If the offering agent makes a profit, the market gains information on the creation of more difficult problems in the domain.

2. If the accepting agent makes a profit, the market gains information about how to effectively solve problems in the domain.

As more and more interactions are produced, more useful information is provided to the market in general, and the individual agents in particular, contributing to the overall knowledge of the problem domain.

2 Market Design

In a classical paper [12], Alvin Roth contends that market design involves the creation of a venue for buyers and sellers and a format for transactions. His main example is that of labor market clearing houses, but he also touches on auction markets for various services. The goal of the market in this sense is to support the mutual satisfaction of buyer and seller preferences. In our case, the design goal of a computational artificial market is to support the constructive peer evaluation of agents, while guaranteeing fair transactions. We use an interpretation of game theory similar to Roth: the computational market includes both human and non-human players (or *teams*). The teams develop and improve their non-human counterparts, the *agents*.

2.1 Algorithmic Markets

Market design is itself complex as described by Roth, though sometimes surprising closed form analysis are possible. In an earlier paper [7], Lieberherr considers the artificial market of boolean constraint satisfaction problems (CSP), showing that for each set of relations there is a *P-optimal* threshold that determines the optimal price for a challenge offered over relations from the set. But, this in no way solves the design problem for this market, since for finite problems the optimal price is often above the P-optimal threshold and the market shifts drastically based on slight changes. With the current state-of-the-art in algorithm design, it is not clear what the optimal prices are for some markets, though there are isolated results like Hastad's result for 3SAT [4] showing that coming closer than $7/8$ to the optimum in polynomial-time would imply that $\mathrm{P} = \mathrm{NP}$.

For a classic CSP based market, classical game theory can be used to approximate the optimal prices as sketched in [8], but market design in general will remain an empirical activity. There are two areas of market designs we wish to explore: those relating solely to algorithm design, and those useful for teaching algorithm design and related objectives.

Algorithm Design: We design market mechanisms where the rules drive developers of the agents towards better algorithms for a certain evaluation objective (*e.g.*, minimizing time and/or space). The agent with the best algorithms, both for providing and for solving hard problems, has the best chance of winning. We expect this experimental approach to algorithm evaluation and analysis will support improvements to the state-of-the-art, pushing the algorithm design and analysis fields forward. A market competition log then becomes a record of possible tests and benchmarks for algorithm evaluation. The agent ranking describes how well the teams understand the problem domain, though the two main components of the agent (providing and solving) can also be judged separately.

Teaching: We design market mechanisms where the rules drive the students to apply principle course objectives (*e.g.*, good software design, or high performance and accuracy). The ranking of agents will correspond to the students' conformance to the instructional objectives, which are linked to a given computational domain. The competition log provides a test suite for students, and frequent contests give teams near constant feedback, allowing them to learn from each other, as well as the course staff.

2.2 Constructive Beliefs

The design of a market is not only closely tied to challenges and the properties of the algorithms involved, but also to what the challenges represent for both the offering and accepting agents. We view offered challenges in the market as representing specific *constructive beliefs*: agents take part in an interactive session where the offering agent attempts to provide a problem that confirms his belief, and the accepting agent attempts to construct a solution that will discount the challenge's underlying belief. Though the profit or loss from a challenge does not necessarily give definitive proof/disproof of a belief, it does provide evidence (for or against) to the quality of the agent's algorithms and likewise the validity of its beliefs.

Behind the design of computational artificial markets is the idea of a *support system* for constructive, mathematical beliefs. For the interaction of two agents, we focus on representative beliefs in a problem domain X, structured as alternating *exists* and *foralls*, the most common being a single pair:

$$\exists x \in X' \,.\, \forall s \in S(x) \,.\, Q(x,\, s)$$

where $X' \subseteq X,\ S(x)$ is a domain of *feasible solutions* for a specific instance of X', and Q is a predicate that describes the essence of the belief. The *belief* itself can be succinctly described by the pair, $\langle\, X',\, Q\rangle$.

The structure of the beliefs around which the market is built allows us to build a protocol for establishing support (or nonsupport) for a belief. This is similar to *interactive proofs* [13], though in our formulation the resources of both participants are limited in some way. Beliefs are *constructive* in the sense that we require that they be supported by constructing instances of problems and solutions. A market for beliefs is interesting because the naïve constructive proof or disproof of the belief is difficult: it would require us to test all instances $x \in X'$, and all its corresponding feasible solutions in $S(x)$.

Specialized markets for different domains may be based on more complicated forms of belief, but the structure of the beliefs establishes rules for agent interactions. The market ensures that the offering and accepting agents coordinate, albeit with different goals, to provide a measure of proof for or against the belief, while the constructive nature of the beliefs lends itself to an algorithmic interpretation for providing and solving problems. One of the research goals of this project is to solidify the mathematical foundations of computational markets, and explore the representation, structure, and proof of the underlying constructive beliefs.

2.3 Beliefs to Challenge Markets

Beliefs represent constructive mathematical claims, but in the end, markets (even artificial ones) require a notion of transactions and profit or loss. In order to create a market, we define it over *challenges*, which are in turn defined as a specific belief at a given *price*. Using our definition of belief above, we define an *offered* challenge as a belief/price pair:

$$\langle \mathit{belief},\, p \rangle \;\equiv\; \langle \langle X',\, Q \rangle,\, p \rangle$$

where p is a real number. The market definition also requires a function, $\mathit{payment}(r,\, p)$, that describes the amount to be paid to the solver based on the price, p, and a boolean solution result r. Again the challenge represents a belief:

$$\exists x \in X' \,.\, \forall s \in S(x) \,.\, Q(x,\, s)$$

and the result, r, is the value of our predicate, Q, describing whether x and s support or discount the belief:

$$r = Q(x,\, s)$$

An artificial market is uniquely defined by X, S, $\mathit{payment}(r,\, p)$, and the *structure* of the underlying beliefs, which respectively represent the problem domain, solution domain, profit calculation, and interaction protocol for challenge transactions. Agents taking part in the market embody two main algorithms: one for selecting a problem instance for an offered and accepted challenge (*provide*) and one for selecting a feasible solution for a given problem (*solve*). In addition, intelligent agents must know (or figure out) which sub-domains to *offer*, which challenges to *accept*, and what prices make challenges attractive, in order to succeed in the market.

2.4 Example

As these are all abstract concepts, it is worth going through a simple example to describe a more concrete computational challenge market. For this we use the domain of *monotone* CSP: constraint satisfaction problems involving only variables without negation. Specifically, our problem domain X will be all structurally correct monotone CSP *formulas*, where a formula is a set 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 a map or function whose domain is $V(x)$, and range is the boolean values, $\{\mathit{true},\ \mathit{false}\}$.

For CSP, the easiest way to define a sub-domain for a challenge (*i.e.*, X') is to do so *intensionally* by giving a set of *relations* that can be used to create constraints. The function $\mathit{fsat}(x, s)$, which calculates the fraction of satisfied constraints in x when using assignment s, is useful within predicate descriptions. Finally, we define a fairly simple *payment* function:

$$\mathit{payment}(r, p) \ = \ \text{if} \ \ r \ \ \text{then} \ \ 0 \ \ \text{else} \ \ 2 \cdot p$$

which repays the solving agent with an overall profit of p if the solution discounts the underlying belief of the challenge, and *zero* otherwise (*i.e.*, a loss of p).

Within our defined CSP market, an agent might offer a challenge using the relation $1\text{-}\textsc{in}\text{-}3$, which is *true* when exactly one of its three variables is assigned *true*. This could be offered at a price of 0.4, represented by the following challenge:

$$\langle\langle\{1\text{-}\textsc{in}\text{-}3\},\ Q\rangle,\ 0.4\rangle \quad \text{where} \quad Q(x,\ s) = (\mathit{fsat}(x,\ s) \leq 0.4)$$

which represents the belief that solutions to a specific CSP instance (to be provided) using only the $1\text{-}\textsc{in}\text{-}3$ relation will not have a fraction of satisfied constraints greater than $2/5$. If accepted, the offering agent would need to provide a CSP instance using the $1\text{-}\textsc{in}\text{-}3$ relation, *e.g.*:

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

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

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

For formulas made only of this relation, $1\text{-}\textsc{in}\text{-}3$, there always exists an assignment that will satisfy at least $4/9$ of all constraints. This is known as the *P-optimal* threshold [7]: we can satisfy this fraction quickly (in polynomial time), but to do better than $4/9 + \epsilon$ with $\epsilon > 0$ for arbitrary $1\text{-}\textsc{in}\text{-}3$ formulas 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 $1\text{-}\textsc{in}\text{-}3$, eventually obtaining the maximum value $4/9$ at $p = 1/3$.

2.5 Secret Challenges

While certain aspects of the computational market may have definite solutions, there are assumptions made with respect to the limits (or lack there of) placed on the agents when providing problems and solutions. Once we place fixed limits on the resources agents may use we find that a new form of market emerges, referred to as the *secret challenge market*, where agents offer challenges that represent a specific secret belief, resource limitation, and price. For simplicity we focus on *time* limitations, though other resource restrictions (such as *space* or *power consumption*) are possible.

The *secret belief* is constructive as before, but we add a *time* limitation, use a three argument *secret* predicate. The predicate itself is not *secret* per se, but the market is required to maintain the security of the secret solution throughout transactions. Although a time limit could be set globally for the market, we include it in the belief for completeness.

$$\mathit{belief}_{sec} \equiv \langle X',\, t,\, Q_{sec} \rangle$$

The secret belief represents a mathematical belief involving a a time limitation and a 'secret solution' for a particular problem. The predicate, Q_{sec}, is then able to compare two feasible solutions to the same problem:

$$\exists x \in X',\, s_{sec} \in S(x)\,.\,\forall s \in S(x)\,.\,\mathit{time}(s,\, x) < t \Rightarrow Q_{sec}(x,\, s_{sec},\, s)$$

The secret challenge market brings us closer to algorithmic comparison, since the offering agent must provide a secret solution in addition to the problem instance. The secret market is interesting because an accepting agent knows that a provided problem is at least solvable, and satisfying the predicate will usually involve achieving a specific approximation ratio to the secret solution, within the given time. If we require that a secret problem/solution be created during the market contest or within a specific time bound, then we are able to definitively compare the algorithms of both agents.

3 Current Research

The idea of using artificial markets for algorithm evaluation and teaching has come out of past and current research in various computational domains. In particular, we have focused on incarnations of market-based games involving constraint satisfaction problems (CSP) with a design that we call the Specker Challenge Game [6] (SCG). SCG itself came out of the *Evergreen Game* [1], a simple two player challenge game where one player provides a CSP formula, and the other attempts to compute an assignment that maximizes the formula's satisfaction ratio. Both players attempt to improve their respective algorithms: the provider to find the most difficult CSP instances, and the solver to find the best solutions quickly.

3.1 Specker Challenge Game

The evergreen game eventually evolved into a multi-player game where players offer and accept challenges, and subsequently provide and solve problem instances. The idea of the game became more general, being parametrized by the domain in which problems (and the market) are defined. For instance, in our most common instantiation, SCG(CSP), each challenge consists of a problem *type* and a *price*. Similar to our 1-IN-3 example, the problem type is a set of *relations*, which, if the challenge is accepted, can then be used to construct a CSP formula for the accepting agent to solve. The price is a real number in the interval $[0, 1]$. SCG challenges represent beliefs similar to our earlier example except that the predicate uses the challenge price for a bound, instead of an arbitrary constant:

$$Q(x,\, s) = (\mathit{fsat}(x,\, s) \leq p)$$

Once accepted, a specific problem instance is provided: a list of *constraints*, each with a relation from the challenge type and a set of variables. The acceptor of the challenge must then solve the problem by constructing an *assignment* to all variables. The *quality*, q, of an assignment s to a formula x (defined with $\mathit{fsat}(x, s)$) is used to define the market's augmented *payment* function:

$$\mathit{payment}(r, q, p) \;=\; q$$

In the educational spirit of the game, an agent always has a chance of making some amount of money back, regardless of whether or not the challenge's belief has been discounted. Overall, an accepting agent makes a profit (or loss) of $quality - price$.

As a complete example, suppose team (or agent) A wishes to offer challenges over formulas using the relations *nand* and *or* at price 0.5. The agent would offer the challenge:

$$\langle\, \{\textsc{Nand}, \textsc{Or}\},\ 0.5\, \rangle$$

If an agent B accepted this challenge, it would first pay the price, 0.5, to A, then A would provide a corresponding problem instance *e.g.* :

$$\langle\, (\textsc{Nand}\,\{a,\, b,\, c\}),\ (\textsc{Or}\,\{a,\, b\}),\ (\textsc{Or}\,\{b,\, c\})\, \rangle$$

Finally, B would construct a complete assignment, *e.g.*:

$$\{\, a \mapsto \mathit{true},\ b \mapsto \mathit{false},\ c \mapsto \mathit{false}\, \}$$

which yields a satisfied fraction of $2/3$ (constraints 1 and 2 are satisfied). Team B would then be reimbursed $2/3$, for an overall profit of $1/6$. This is, of course, a toy example since SCG agents typically produce formulas with thousands of constraints. We have experimented with different payment functions, though this seems to be the easiest scheme for students (and their agents) to work with and understand.

Following our description of artificial markets, an SCG *contest* is made up of a number of *rounds*, in which each team gets a chance to offer new challenges, accept other's challenges, provide problems, and solve previously accepted challenges for which problems have been provided. The contests are run automatically by a trusted administrator (a software component) that imposes a number of rules in order to keep the game fair and interesting. For instance, making sure agents provide and solve problems as soon as possible (in the next round), and enforcing that teams accept a minimum number of challenges in each round or *re-offer* them at prices they find more reasonable. We have recently adapted our administrator and initial agent to interface over the Internet (using HTTP), which allows us to run frequent, more distributed competitions.

3.2 Theory in SCG

The artificial market of SCG(CSP) becomes more interesting when we realize that agents can make educated decisions for offering and accepting challenges, as well as providing and solving problem instances. As with our 1-IN-3 example, for the SCG *payment* formulation there is an algorithm for calculating the *break-even* price based on the *type* of a challenge, which is derived from the truth table of its corresponding relations. The process can become quite complicated for large sets of relations, but it involves a *polynomial* that describes the probabilities of satisfaction for infinite formulas. Once the break-even price has been calculated for a set of relations, it can be used as a strict lower bound offering price, or a rough upper bound accepting price for challenges over those relations. Maximizing the polynomial also yields a *maximum-bias* that gives accepting agents a method for constructing a near optimal random assignment efficiently, and even an optimum *blend* of the relations in order to provide the most difficult problems.

In the case of 1-IN-3 the break-even price is the same as its P-optimal threshold: $4/9$, or $0.4\bar{4}$. For our example involving the relations NAND and OR, the break-even price is approximately 0.815. Knowing this price helps the offering agent select a more profitable price (above the break-even), but knowing the *maximum-bias* gives the accepting agent a very good starting point for better solving a problem instance. As the relation name implies, the maximum-bias for 1-IN-3 is $1/3$. For formulas made up of NAND and `Or` the maximum-bias is closer to even, at approximately 0.57. Of course, this does not solve the problem of the SCG market since completely random assignments often give decent results, though it does put the agents on a more level playing field. Teams can then focus on more detailed algorithms for solving and providing more difficult problem instances. The details of polynomial derivations, break-even prices, and maximum-bias can be found in [7], as well as randomized and derandomized algorithms for calculating them using well known techniques by Spencer and Erdös.

3.3 Teaching and Tools

Implementations of SCG(CSP) have been used at Northeastern University to teach several semesters of a senior level course on software development [5]. Throughout the course students learn not only about the rigors of software organization and evolution, but also about the details of SAT/CSP solvers, algorithms, and performance. The feedback from both course staff and regular competitions keeps students involved, putting their best ideas into the agents.

In order to support the development of various SCG components including the market administrator and agents, we have built up a number of extremely useful software tools. Our research has focused on creating a generative development tool and library, called DemeterF [3], which supports a more generic style of development in an object-oriented paradigm, using the Java and C# programming languages. The generic nature of our tool has allowed us to begin work on a more general form of SCG that can be parametrized by the problem domain and market protocol, which will be the study of further research.

4 Proposed Research

We propose to study the design of computational artificial markets, the theory behind them, and develop tools and techniques to support the implementation of agents and market contests. In this section we explore the interesting questions in each space, and provide hints as to the most likely directions of our research.

Design: We propose to study the design space for computational artificial markets with specific objectives.

The market must ultimately be about driving agents and teams to improve their algorithms for providing and solving difficult problems, but this relies in part on the willingness of agents to interact. Currently in SCG, agents are driven to engage in each round of a market contest by forcing them to offer and accept at least one challenge. If an agent decides that none of the challenge prices in the market are attractive, then it is required to reoffer all challenges at cheaper prices. We would like to explore other ways to engage agents in the market, for instance, by simply forcing them to accept challenges (eliminate our reoffer rule), or requiring that an agent pay a participation fee for each round.

In the implementation of a market, the interactions (protocols, datatypes, *etc.*) between the market administrator and the individual agents is defined through a parametrized interface. How can the protocol/interface parameter be defined so that a non-trivial market results? Is it possible to develop a design for particular instantiations that maintains the desired overall market properties? In particular, we wish to study whether or not this protocol/interface can be designed to be automatically derivable from the problem domain and the structure of the market's underlying beliefs, and investigate various languages for describing the necessary market components.

The design of both useful and interesting belief predicates for a domain is itself an interesting problem. Can we infer interesting belief predicates from a description of the problem domain, and allow the market to decide the most useful predicate types? Or is it best to use the design of the predicate language to shape the goals of the market contests?

The interactions of agents is based on trust in the market; if a contest is not fair, there will be limited interest in participation. Trust in the market is destroyed if it is possible for agents to violate market rules while avoiding detection. The drive for an agent to demonstrate better algorithms will naturally lead agents to focus on problems that other agents find difficult, but all interaction must be within the laws of the market. Security is currently overseen and enforced by the market administrator, but there are a number of possibilities for distributing the control. By introducing more secure protocols and other self-checking forms of interactions, we can possibly reduce the amount blind trust that agents must place in the administrator.

In an artificial market, knowledge that a team or agent has about the algorithms of other agents will likely change based on the results of a market contest, as recorded in the market's log. In our formulation of SCG, the market history is published after a contest so that all players may learn from everyone's successes and failures. While this works for an open educational enterprise, it may not be feasible for confidential or proprietary agents. It is worth exploring whether or not (and how much) the loss of information slows algorithmic progress in the market.

Theory: We propose to develop a theory to understand computational artificial markets.

A series of market contests can be viewed as a repeated, *zero-sum* game. In mechanism design for games, the designer chooses the structure of the game and is interested in its outcome. The mechanism of SCG guides teams (and agents) to what is ultimately a common good: to find better algorithms to solve and provide the most difficult problems. This kind of mechanism design is different from typical mechanism design studied in Game Theory [9, 11], but we cannot be sure whether general mechanism design principles still apply. We wish to see which computational markets can be analyzed, whether or not they have equilibria, and if so, how these can be computed.

A *Bayesian* game is one in which information about certain characteristics of other players is incomplete. A computational market contest is certainly a form of Bayesian game, but one in which an agent's knowledge about others' algorithms is incomplete. This level of abstraction is above the current state of the art in Bayesian games, but is nonetheless a theoretically interesting problem.

Our SCG formulations have focused on CSP and the theoretical foundations of P-optimal thresholds for different challenge types. Are there similar break-even prices or thresholds for other computational markets? How much information can an accepting agent really gain from the description of a challenge's sub-domain and a belief's predicate?

Tools: We propose to design, develop, and distribute useful software development tools and techniques.

The creation and running of artificial market contests makes heavy use of both generative and generic tools in order to create a specialized market instantiation for a given problem domain. This includes automatically generating a trusted market administrator and an initial agent for teams; determining data structures and communication mechanisms; and establishing the interaction protocol for agents. New development tools and techniques are required to allow particular markets to be created quickly with limited human involvement. In general this will be done through the creation of domain specific languages and tools, which can then be distributed for the better of the broader software development community.

We have developed a state of the art generative and generic tool, called DemeterF [3], that we currently use to develop our administrator and agents. We propose to use artificial markets as a challenging application to drive the further development of DemeterF and other tools. DemeterF is the latest incarnation of Adaptive Programming which was initially developed with NSF support including various tools and ideas that are now widely used in Internet/XML-based technologies. Our recent work on evolution of generic programs [2] will help keep agents flexible in the face of dynamic markets.

Each market competition produces a large log of history information: challenges offered and accepted, problems provided and solved, *etc.*. Various data mining tools and techniques are needed to search this history and discover how money was made and lost on different challenges. As support for our design and theoretical claims it is also necessary to recover the most difficult problems for *offline* agent testing and evaluation. We also propose to further develop and refine our tools based on insights from the design and theory described above.

Ultimately, our research progress will be aided by feedback from staff, students, and the wider theoretic, algorithmic, software development, and user communities. We see each of the research areas as providing interesting insights to be used in other areas, in addition to contributing to the overall goals of the project.

5 Curriculum Development Activities

Since Fall 2007, we have been using artificial markets to teach computer science using a special formulation of SCG(CSP) in both graduate and undergraduate courses. Each course level invites students to master a wide variety of computer science topics including theory of computation, discrete structures (combinations, permutations, and simple combinatorics), Calculus (maximizing and minimizing multi-variable rational functions), object-oriented design, and software engineering. An important part of teaching is to be able to evaluate students as to whether or not they understand the material and whether they can effectively use the ideas in practice. A market contest with agents developed by student teams is the perfect tool to provide feedback on how their understanding of the necessary algorithms compares to other students, and the agents provide a more interesting testbed for student ideas.

When students enter in the course, after an introduction they receive the source code of an initial agent for a particular computational market. The initial agent has a minimum amount of built in logic, but the students are required to add much more intelligence in order for it to succeed in the market. We received reports from students in Spring 2009 that they became completely immersed in the development of their agent and improving their algorithms. In order to give students a reward for implementing their agent's intelligence, while still allowing all students to learn from each other, all agents' source code was revealed two weeks after a contest was run. This led to the rapid improvements of all the agents, but also contributed to the competitive nature of the market. One of the main goals of this proposal is to document and further develop our use of artificial markets for teaching in computer science. There seems to be significant interest from students when starting with a *baby* agent that they must nurture into an intelligent *adult*.

We also propose to polish our tools and techniques to support multi-university competitions, leveraging the distributed nature of the Internet to allow a common computational market to be inhabited by agents as geographically dispersed as possible. Once universities decide on a computational domain, students from different states and countries can compete in weekly (or even daily) contests, giving the participating teams immediate feedback on the success of their agent. Of course, the learning is not limited to students, but also extends to faculty and researchers at various institutions who can test and evaluate new algorithmic techniques in the common market, allowing faculty, scientists, and students to remain involved.

6 Plan of Work

6.1 Year 1

- Study the theory of mechanism design for the design of algorithmic markets and their relatives in game theory.
- Complete the implementation of the parametrized generic market. It is important that the agent contests are completely automatic and run at specified intervals over the Internet,*i.e.*, that the competitions not be time consuming for the agents or the teams involved. Automatic contests require a trustworthy administrator that enforces the necessary rules and provides informative error messages when agents break the rules.

  All this must to be done in the presence of a complex parameter: the chosen computational domain. The design of general parametrized market tools involves the design of languages for expressing the computational problems, their solutions, the beliefs, challenges descriptions, and the payment functions.
- Support decision procedures as problem domains and design a market for decision procedures (we have been focused on computational optimization problems).
- Support various general computational processes in order to allow the teaching a variety of computational domains, *e.g.*, biological and physical systems.
- Document all software and publish interfaces. Prepare a web page that describes our Artificial Market for sharing and evaluating computer science knowledge and distribute it to the educational and algorithmic research communities. A very exciting aspect of our computational market is that it proposes a new way to evaluate algorithms. Both the quality of solution algorithms and asymptotic performance are important, but eventually a practical implementation has to compete effectively on a range of problem sizes.
- Make the tools (administrator and initial agent generators) available on line. This will require that we publish enough information so that members from outside the research group can become market designers, with the help of our tools.

6.2 Year 2

- Organize distributed competitions over the Internet. In general, competition frequency is only limited by the amount of information teams can gain from the contests, and the number of teams involved.

  Competitions will be in 3 general categories:

  1. Beginning algorithm students (advanced undergraduates). At this level we have already accumulated significant experience [5]. Contests can likely be run more frequently than other categories.
  2. Knowledgeable algorithm designers (Masters students and beginning PhD students). Working in a variety of computational domains.

3. Experienced algorithm designers (researchers and advanced PhD students). Working in specialized, complex problem domains.

- Develop data mining tools for contests to make it easy for the teams to learn from their agent's market experience. This involves recovering tests, benchmarks, and performance information for agents, challenges, and problem instances.
- Receive contest feedback and improve our designs/tools.

6.3 Year 3

Complete the balance of the tasks described in Section 4 that were not not completed within years 1 and 2.

7 Results from Prior NSF Support

At Princeton University, Dr. Lieberherr received two NSF awards: MCS80-04490 and MCS82-01878 on Combinatorial Optimization and Search Problems which laid the foundation for the artificial market *SCG*. Some of the SCG foundational work is mentioned in the Biographical Sketch section.

References

[1] A. Abdelmeged, C. D. Hang, D. Rinehart, and K. J. Lieberherr. The Evergreen Game: The Promise of Polynomials to Boost Boolean MAX-CSP Solvers. Technical Report NU-CCIS-07-03, Northeastern University, April 2007.

[2] A. Abdelmeged, T. Skotiniotis, and K. Lieberherr. Controlled Evolution of Adaptive Programs. In *IWPSE-EVOL 2009, Workshop on Principles of Software Evolution*, 2009. `http://www.ccs.neu.edu/~lieber/papers/contr-evol-ap/iwpse32-abdelmeged1.pdf` .

[3] B. Chadwick. DemeterF: The functional adaptive programming library. Website, 2009. `http://www.ccs.neu.edu/~chadwick/demeterf/` .

[4] J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. `http://portal.acm.org/citation.cfm?doid=502090.502098#` .

[5] K. Lieberherr. Software Development: CSU 670 Spring 2009. Website, 2009. `http://www.ccs.neu.edu/~lieber/courses/csu670/sp09/csu670-sp09.html`.

[6] K. Lieberherr. The Specker Challenge Game. Website, 2009. `http://www.ccs.neu.edu/~lieber/evergreen/specker/scg-home.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 S. Vavasis. Analysis of polynomial approximation algorithms for constraint expressions. *Lecture Notes in Computer Science*, 145:187–197, 1983. 6th GI-Conference Dortmund, January 5-7, `http://www.ccs.neu.edu/research/demeter/papers/vavasis/PolyAppAlg-Vavasis.pdf`.

[9] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[10] C. Papadimitriou. Algorithms, games, and the internet. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 749–753, New York, NY, USA, 2001. ACM.

[11] D. Parkes. When analysis fails: Heuristic mechanism design via self-correcting procedures. In *SOFSEM 2009: Theory and Practice of Computer Science*, volume 5404/2009, pages 62–66. Springer Berlin / Heidelberg, January 2009.

[12] A. E. Roth. Game theory as a tool for market design. In *Game Practice: Contributions from Applied Game Theory*, pages 7–18. Kluwer, 2000. `http://kuznets.fas.harvard.edu/~aroth/design.pdf` .

[13] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, December 1996.