Volunteer Computing Using Casual Games

Charles Cusack
Department of Computer Science
Hope College
cusack@hope.edu

Chris Martens
Department of Computer Science
Carnegie Mellon University
cmartens@andrew.cmu.edu

Priyanshu Mutreja
Department of Electrical and Computer Engineering
Valparaiso University
priyanshu.mutreja@gmail.com

Abstract

We introduce the idea of volunteer computing games– *that is, casual games which are implementations of distributed algorithms. Volunteer computing (VC) is a form of distributed computing which seeks to harness the computational power of individuals from around the world for free. Although the numbers and types of VC projects has grown significantly over the past decade, the majority of participants in these projects are still from a limited demographic, and there are still many people who know nothing about these projects. On the other hand, most people know about casual games, and a majority of people play them. We propose that the use of casual gaming in volunteer computing projects can significantly increase participation, and therefore success, and we describe a prototype of a game that solves the maximum clique problem.*

1 Introduction

Teams of scientists, engineers, and mathematicians around the world are trying to solve many difficult problems through the use of computation. For some of these problems, the computational power of a single processor is inadequate for the task. Distributed computing permits researchers to spread their computation across multiple processors, allowing much more computation in a shorter period of time. However, even when access to a distributed computing environment is available, there is a limit to the number of processors available.

In light of this, there have been many efforts over the past decade to encourage users around the world to volunteer their spare computing cycles to solve these problems. Although the exact form of the distribution and the terms used to describe it have varied (volunteer computing, global computing, internet computing, web-based metacomputing, etc.), the concept is the same: get others to help do the work. Some notable examples of volunteer computing include GIMPS (the Great Internet Mersenne Prime Search) [8], Distributed.net [6], SETI@Home [22], and, more recently, Berkeley Open Infrastructure for Network Computing (BOINC) [3].

Another class of problems of interest is problems which computers are either unable to or are very poor at solving. For instance, computers are not currently very good at labeling images. More recently, the concept of distributed *human* computing has become popular as a solution to this problem. Here the interest is in getting the volunteers themselves, and not their computers, to help in computations. Some of the most notable examples of this are Luis von Ahn's web-based games that seek to label images [7, 19, 20], Wikipedia [31] and other Wikis, and Amazon Mechanical Turk [2, 24].

For an extensive list of distributed computing projects on the web, see [5], and for a thorough introduction to VC, see [21].

2 Motivation

2.1 Barriers to Volunteer Computing

Obviously there are many projects which seek to use the computational power of computers and their users across the world. What is surprising is how many

projects there currently are, and how (relatively) few people actually know about them. Further, of those who know about the projects, few actually participate. An open problem seems to be how to get more people to participate. There are several barriers to user participation in VC.

The first barrier is lack of awareness. It does not matter how much computing power people have, or how interested in a project they might be, if they do not know the project exists. Many of the current projects suffer simply because the majority of people have never heard of them.

The second barrier is lack of broad appeal. According to a recent survey, the top two reasons for participation are "contributing to scientific research" and "contributing to statistics/friendly competition." In fact, it seems that the concept of VC itself, especially the co-opetition (cooperative competition) involved, appeals to participants more than the particulars of the projects. Participants are more interested in increasing their ranking than furthering science, and the most significant contributions to most projects are from individuals and teams whose goal is to top the leader board. [9]

The third barrier is limited demographic. According to an ongoing survey on the BOINC website [4] with over 7400 responses currently, 96.5% of participants in BOINC projects are male, 5% are under the age of 19, 50% are between the ages of 20 and 39, 38% are between the ages of 40 and 59, and 7% are over 60. In terms of computer expertise, 63% consider themselves advanced, 35% intermediate, and only 2% beginners. Another recent survey has similar findings [9], making it reasonable to assume that these statistics are similar for most VC projects. What this tells us is that most participants are middle aged males with a significant amount of computer expertise. Almost entirely absent from participation are women, children, the elderly, and those with limited computer expertise.

The fourth barrier is lack of technical savvy. Many projects require users to download and install special software, something that can be difficult for a large segment of the population. We have already seen that people who consider themselves "beginners" in using computers do not generally participate in VC, and this may be one of several reasons. Some users think that the more applications that are installed (but not necessarily running) on their computer, the slower their computer is. Others are afraid that these programs might contain spyware or other malicious software.

2.2 Casual Gaming

Casual games are those games that are designed to be played with a minimal amount of instruction and/or skill and in a small amount of time (like during lunch or before the boss walks by). The forms of the games vary, and include card games (various versions of Solitaire and Poker, etc.), puzzle games (Bejeweled, Luxor), action games (Diner Dash, Mother Load), and strategy games (Oasis, Tribal Trouble).

The casual gaming industry, particularly casual gaming on the web, is booming [11, 13], and there is every indication that it will continue to grow. At any given time there are millions of users playing games on one of dozens of online gaming sites like Yahoo! Games, RealArcade, and MiniClip, to name just a few (See pages 16-19 of [12] for a more complete list). It is believed that in the US alone, over 100 million people will play a video game this year [12]. There are companies that specialize in developing, publishing, and distributing casual games. Given the amount of advertising found on these gaming sites and within the games themselves, it is clear that many companies believe these game get sufficient exposure to warrant the expenditure of funds.

Earlier this year, Macrovision released results from a survey of 789 participants who play casual games on trygames.com [14]. The study revealed that 71% of players are female, 37% of players are between the ages of 35 and 49, and 28% are between 50 and 60. The *typical* casual gamer is a woman in her forties [12]. 37% play 9 or more game 'sessions' per week, with 66% saying their sessions last at least one hour, and 31% saying their sessions last at least two hours. 67% play puzzle games, 44% play card games, 35% play strategy games, and 34% play action games.

2.3 Breaking Barriers

We believe that all of the barriers that hinder VC can be broken through the use of casual games.

Most people are totally unaware of the concept of VC, let alone specific projects. However, almost everybody knows what casual games are, and many people play them. Thus, if VC projects could be implemented as games and either published on established gaming sites, or if a new *volunteer computing games* site were developed and promoted, the awareness barrier could be broken.

If a project is implemented as an interesting game, it will attract not only people who are interested in the project itself, but also those who enjoy playing the game, regardless of whether or not they care about the project. Thus, the appeal barrier can be broken.

Since it has been widely established that men and women of almost all ages play casual games, the demographic barrier can be broken. Further, since casual games require no special skills to play, and often do not require installation, the fourth barrier is broken.

In addition to breaking these barriers, a melding of participation can occur–a volunteer game will attract both those users who are interested in VC or the project itself, and those who are interested in the game. It will attract those who want to climb up the leaderboard of the game, and those who want to climb up the leaderboard of the computation. It will attract those who want a temporary diversion from the world, and those who want to help save the world. In fact, the proper game design can allow the user to save the world in the game as they help save the real world.

Finally, unlikely as it may seem, it may be possible to get users to pay to perform computations, since if a game is good enough, they might be willing to pay to play it, generating revenue to continue and expand the project.

2.4 Previous Work

This approach has already been successfully used in distributed human computing projects, with the most notable examples being the ESP Game [7, 27], Peekaboom [19, 30], and Phetch [20, 28], all games where players work together to accomplish what computers cannot do well: labeling and describing images. This is done in the context of games so that the users do not feel like they are simply "labeling images." The game Verbosity [26, 29] seeks to collect common-sense facts, which is another task that is easier for humans to do than computers.

2.5 Moving Forward

To our knowledge, the use of casual games to solve problems that are more computational in nature has not been attempted. We outline four possible approaches of implementing a VC problem as a casual game, and discuss their relative merits.

The first approach is to find a way to turn traditional distributed algorithms into games, and let lots of people play the games so that eventually they will have run the algorithm on all of the data. The weakness of this approach is that for most computational problems, humans are thousands, if not millions, of times slower than a computer. Thus, it takes the effort of thousands of participants and their computers to do what just one computer can do. Clearly this approach is not practical.

The second approach is to supplement the first approach with an option for the user to also run the "game" in the background[1]. This has the advantage that we get the computational power of the user's computer, and as long as the game is enjoyable, they will continue to allow the algorithm to run in the background as well. In fact, the user does not even need to be playing the whole time–the background process can be run for as long as the user is willing to let it.

The third approach is like the second, except that an attempt is made to develop games which not only allow the user to run our algorithm, but also allow the user to import their insight into the problem. In this way we could have many users thinking about how to make better choices, and a solution might be found because one user had some insight that we did not. This has all of the advantages of the previous approaches, with the added possibility of an extra boost of human insight.

The fourth approach is to implement *any* game, and require the user to run the algorithm in order to play the game. In other words, the user is paying to play the game with their CPU time. This has the advantage that the game design and algorithm design can be separated, allowing for the use of different sorts of games to attract different sorts of users.

2.6 Game Design

It is obvious that the success of any of these approaches depends heavily on the design of the games. If the games are not fun, people will not play them. If they do not have a wide appeal, the demographic barrier will remain.

As an example, the game MudCraft [16] is a real-time strategy game that was specifically designed for the "casual, mixed-gender audience" [32] by utilizing research into gender issues in game design. According to results from their playtesting, they succeeded in

[1]This is an option because if the algorithm was just automatically run in the background, we would really be doing *stealth computing*, and there might be strong objection to this.

their goals. In order for volunteer computing games to succeed, the same care must be taken in game design.

Three key components in the design of a game are mechanics, dynamics, and aesthetics [10]. The mechanics refer to the rules of the game. In the context of VC games, the mechanics are very restricted. That is, we will generally not be able to make up whatever rules we want. Given a problem and perhaps one or more algorithms to solve the problem, there will be specific rules that must be followed in order for the game to correctly solve the problem and/or execute the algorithm.

The dynamics of a game refer to the run-time behavior of the game, and are a direct result of the mechanics. Two different games might have the same mechanics but very different dynamics, making them appear to be different. This can be exploited to attract different sorts of users to solve the same problem.

Finally the aesthetics of the game are its "look and feel." This includes things like the theme (ancient Egypt, outer space, the old West, etc.), story (saving the world, rescuing the damsel in distress, etc.), graphics (based on the theme), and sound. Again, this is an area that can be tweaked to attract a larger and more diverse audience.

2.7 Applicability

There are a range of problems that are being solved using volunteer computing. On the one extreme there are number-crunching projects that are best solved by computers (like SETI@Home and GIMPS). On the other extreme there are problems, like image labeling and common-sense fact gathering, that are easily done by humans, but either impossible or extremely difficult for computers.

Although volunteer computing games have been successfully used for the latter group of problems, it may not be as easily applicable to the former group[2]. For instance, we highly doubt that a game can be implemented to assist in the SETI@Home or the GIMPS projects. However, we believe that there are a large number of problems that lie in between these extremes that can be successfully solved using games.

Next we will demonstrate that it is possible to solve certain types of searching problems using a simple game of choice.

[2]Unless the fourth approach, using unrelated games, is used.

3 A Volunteer Computing Game

We now present our implementation of a simple "game" that acts as a distributed algorithm to solve the MAXIMUM CLIQUE PROBLEM. We focus our discussion on the MAXIMUM CLIQUE PROBLEM for several reasons. First, a recent parallel algorithm to solve this problem was the basis of our volunteer algorithm. Second, as one of the better-known examples of an NP-Complete problem, instances of it are often used as benchmarks for parallel algorithms. Finally, by restricting to this specific problem, we can more clearly describe how the algorithm is turned into a game.

It is important to realize that our game can be adapted to solve many problems for which backtracking algorithms are known.

3.1 Maximum Clique Problem

A *graph* is a pair (V, E), where V is a set of *vertices*, and E is a set of unordered pairs of vertices, called *edges*. A *complete graph* is a graph in which every pair of vertices is connected by an edge. Figure 1 shows a complete graph with 5 vertices.

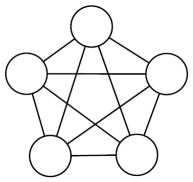

Figure 1: A complete graph with 5 vertices

A *subgraph* of a graph is a pair (V', E') such that $V' \subseteq V$ and $E' \subseteq E$. A *clique* is a subgraph of a graph G that is a complete graph, and *maximum clique* is a clique of largest size in G. The MAXIMUM CLIQUE PROBLEM, as the name suggests, is to find the maximum clique in a graph.

Figure 2a shows a sample graph on 6 vertices, and Figure 2b shows the maximum clique in the graph. In this case, the maximum clique has size 4, and it is unique. In general, it is possible for a graph to have multiple maximum cliques.

3.2 Solving Maximum Clique

The MAXIMUM CLIQUE PROBLEM is known to be NP-complete, so there is no known polynomial-time algorithm to solve it. However, there has been much effort to find algorithms that are fast enough for problems of

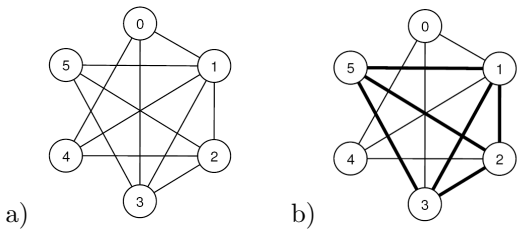

Figure 2: a) A sample graph. b) The same graph with the maximum clique in bold.

reasonable size [1, 17]. In recent years, several parallel algorithms for solving the maximum clique problem have been suggested [18, 23].

In recent months, Thimm, et. al. [23] proposed a parallel algorithm for the maximum clique problem that uses *stealstacks* to assist in load balancing. It turns out that by replacing the stealstack idea with a centralized database, the algorithm can be adapted for use in a volunteer computing setting. We will briefly describe the serial version of the algorithm, how to parallelize it, and how it can be turned into a game.

The first step of the serial algorithm is to order the vertices. Although there may be advantages to certain orderings, we will assume an arbitrary ordering. Next, for each vertex x, $x.Adj()$ denotes the list of vertices adjacent to x, and $x.Larger()$ denotes the set of vertices that come after x in the ordering.

The algorithm traverses a search tree based on the value in each *node* of the tree. A node (C, X) consists of two piece of information– the *current clique* (C), and the *extension set* (X). C is the ordered list of nodes which form a clique that we are currently trying to extend to a maximum clique. X is the ordered list of all vertices which are larger than all of the vertices in C, and are connected to all vertices in C. In other words, X is the set of vertices such that any one can be added to C to extend it to a larger clique.

We start with an empty stack S, an empty list of vertices, M, which will store the vertices of the maximum clique, $C = \{\}$, and $X = V$. So there is no confusion, when we perform the operation $S.Push(C, X)$, we assume that copies of C and X are stored on the stack. The algorithm is given in Figure 3.

Figure 4 shows the search tree that is traversed for the graph from Figure 2a. Each node is labeled with the current clique C and extension set X in the format "$[C|X]$," except leaf nodes, which are drawn in gray without extension sets (since the extension set is empty).

```
MaximumClique(C,X)
    while(X.notEmpty())
        x = X.getFirst()
        X.remove(x)
        if (X.notEmpty())
            S.Push(C,X)
        C.insert(x)
        if (C.Size() > M.Size())
            M=C
        X = intersect(X, x.Adj(), x.Larger())
        if (X.empty())
            (C,X) = S.Pop()
    return M
```

Figure 3: The Maximum Clique Algorithm

Let us demonstrate a few steps of the algorithm for this example. We start by setting $x = 0$, and we set $X = \{1, 2, 3, 4, 5\}$, and push $(\{\}, \{1, 2, 3, 4, 5\})$ onto the stack. Next, we set $C = \{0\}$, and since it is larger than $M = \{\}$, we set $M = \{0\}$. Next, we set $X = \{1, 2, 3, 4, 5\} \cap \{1, 2, 3, 4, 5\} \cap \{1, 3, 4\} = \{1, 3, 4\}$. Since X is not empty, we continue at step 1.

We set $x = 1$, $X = \{3, 4\}$, and push $(\{0\}, \{3, 4\})$ onto the stack. We then set $C = \{0, 1\}$, $M = \{0, 1\}$, and $X = \{3, 4\} \cap \{2, 3, 4, 5\} \cap \{0, 2, 3, 4, 5\} = \{3, 4\}$. The stack now has $(\{\}, \{1, 2, 3, 4, 5\})$ and $(\{0\}, \{3, 4\})$ on it.

Again we continue at step 1, where we set $x = 3$, $X = \{4\}$, and push $(\{0, 1\}, \{4\})$ onto the stack, and set

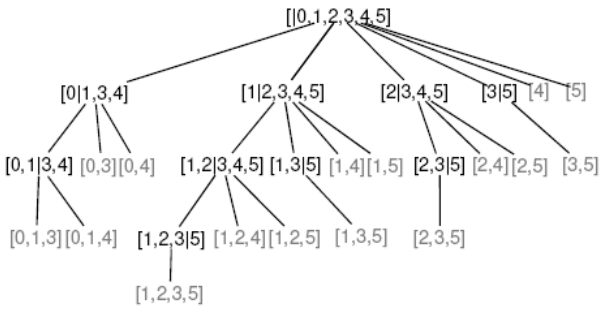

Figure 4: The search tree for the graph from Figure 2a

$C = \{0, 1, 3\}$, $M = \{0, 1, 3\}$, and $X = \{4\} \cap \{4, 5, 6\} \cap \{1, 2, 5\} = \{\}$. Since X is now empty, we pop the stack, so $C = \{0, 1\}$, and $X = \{4\}$ (that is, we take off what we just pushed on), and we continue to step 1. The stack now has $(\{\}, \{1, 2, 3, 4, 5\})$ and $(\{0\}, \{3, 4\})$ on it.

Again at step one we set $x = 4$, and $X = \{\}$. Since X is empty, we do not push anything onto the stack. We set $C = \{0, 1, 4\}$, do not modify M since it also has size 3, and set $X = \{\} \cap \{5\} \cap \{1, 2, 3\} = \{\}$. Since X is empty, we pop, setting $C = \{0\}$, and $X = \{3, 4\}$. The stack now has $(\{\}, \{1, 2, 3, 4, 5\})$.

This continues until we exhaust all nodes on the stack, and eventually get the answer $\{1, 2, 3, 5\}$.

3.3 Volunteer Algorithm

It is actually pretty easy to implement this algorithm in a volunteer computing environment. We start by assuming we have a database on the server which contains the root node of the search tree (That is, the empty solution). When the first client wants an instance of a problem to solve, it receives that node (being the only node available). The first thing the client needs to do is determine if the problem is too large for them to tackle. There are a few ways of doing this. The method used by Thimm et. al. is to use some bound on the number of nodes in the extension set, since generally the more nodes in the extension set, the larger the subtree will be. We use a Monte Carlo algorithm to estimate the size of the subtree.

If the client estimates that the size of the problem they received is too large, the client must *split* the node. To split a node, the client simply creates all of the children nodes of the current node in the search tree, and sends them back to the server so other clients can request them, keeping one to solve itself. This process continues until the client has a node which is small enough. Notice that the tree is split in such a way that once all of the nodes have been solved, the entire tree has been searched, although generally in a different order than if it had been searched with the serial algorithm.

In our previous example, the client would split the root node $(\{\}, \{0, 1, 2, 3, 4, 5\})$ into the 6 nodes $(\{0\}, \{1, 3, 4\})$, $(\{1\}, \{2, 3, 4, 5\})$, $(\{2\}, \{3, 4, 5\})$, $(\{3\}, \{5\})$, $(\{4\}, \{\})$, and $(\{5\}, \{\})$, returning all but one to the server for other clients to solve. If the client kept node $(\{2\}, \{3, 4, 5\})$ and decided it was too large, it would split into the 3 nodes $(\{2, 3\}, \{5\})$, $(\{2, 4\}, \{\})$, and $(\{2, 5\}, \{\})$, sending two to the server, and keeping one to solve.[3]

3.4 A Simple Game

Since the algorithm is essentially a backtracking algorithm, it can be turned into a game quite easily. The regular algorithm proceeds by always picking the first available node to add to the current clique, backtracking if there is nothing in the extension set. In the game version, the user is allowed to pick whichever node they want, and the algorithm proceeds from there. Thus, the only change that takes place is that we replace the first step of the algorithm with a different selection method. This results in only changing the order in which the tree is traversed. When all choices are exhausted, it backtracks automatically, taking the user back to another node where there are still choices to be made.

Once a subtree (rooted at the node they received from the database) has been exhausted, the user receives points based on how many nodes they traversed. Thus, their ranking in the game and in the project can be one in the same. Our implementation includes both the game version and the traditional version of the algorithm, which they may run in the background if they wish (on a different set of data). They receive points whether or not they play the game or run the background process (or both). The high scores show users how they rank against others in their contributions.

Since the background process will generate points at a significantly higher rate, we are exploring how to score in such a way that we encourage participants to keep playing and to keep the background process running. We may score the game separately from the background process, so that participants can be ranked on two lists. This may encourage them to play the game more, which may encourage them to also run the algorithm more. In other words, ranking high in one category may encourage them to try to rank high in the other, which will result in them running the algorithm even more.

Clearly this does not sound like a very exciting game. However, at their core, many popular casual games are not that exciting, either. Although the mechanics sound boring, with the proper dynamics and aesthetics, we may be able to make this into a more interest-

[3] Obviously no reasonable client would actually split any of these nodes, since the problem is very small already.

ing game.

3.5 Implementation Details

The goal of this paper is simply to demonstrate the feasibility of using a game to implement an algorithm. Therefore we have not gone into some important details, including methods of pruning the search tree to speed up the search, security measures, technical details, etc. We finish this section with just a few details for the interested reader.

The design and implementation of our game was driven by three important requirements:

1. No technical knowledge is required of players.
2. Deploying the game requires minimal effort.
3. Extending the game to solve other problems is straightforward.

There are three parts of the game–the client, the server, and the database. The client, a Java applet, communicates with the server, a PHP script, which communicates with a MySQL database. All of these are freely available technologies. Java was chosen for the client because almost everybody has the Java Plugin installed on their computer, so playing the game is as easy as going to a webpage, which fulfills the first requirement.

PHP was chosen on the server side because it does not require a special server application to be running. Because of this, the game can be deployed simply by copying some files onto a machine which has PHP and a web server installed, fulfilling the second requirement. PHP was also chosen because other technologies, like Flash or Shockwave, be be used for clients without modifying the server.

The algorithm upon which the game is based is actually a generic backtracking algorithm. Virtually any problem for which there exists a backtracking algorithm can be implemented by supplying implementation details for a few methods, so the third requirement is met.

4 Future Work

Although we have succeeded in implementing one volunteer computing game for one class of problems, there is still much work to be done before we can judge the validity of volunteer computing games. For one thing, we have not tested our game in terms of efficiency. We know that it is less efficient than traditional volunteer computing algorithms would be (since humans are involved), but we also know that if we could recruit a higher number of volunteers, this would be offset. Unfortunately, we also do not know how many users would play this or similar games. We intend to continue developing our game to the point where we can test and deploy it, hopefully generating some useful data.

As we have stated, our game is not that interesting. Since this project is still in its infancy, we have not yet spent much time thinking about the dynamics or aesthetics of games. It is our belief that even the simplest mechanics can yield an interesting, and hopefully popular, game, if the dynamics and aesthetics are done properly. We hope to partner with individuals in the gaming industry to draw on their expertise in these areas.

There are many other problems and algorithms to consider as candidates for games. We need to explore some of these and figure out a sound approach to converting an algorithm into a game. Perhaps the definition of a problem is enough to develop a game, allowing players to use their own algorithms to play the game.

Finally, the concept of Massively Multi-Player (MMP) gaming has been used to implement a physics classroom [15], and the MMP web-based Urban Dead [25] has turned out to be an interesting social experiment. We have a few thoughts about how we might implement a MMP game that would allow users to interact with each other to solve certain types of problems. For instance, it may be possible to create a scenario where the users are vertices in a graph, and they interact in certain ways to solve graph problems. We hope to explore this idea more in the future.

5 Acknowledgments

This work was funded by the National Science Foundation (NSF) grant CNS-0353566.

References

[1] J. Abello, P. Pardalos, and M. Resende. On maximum clique problems in very large graphs. In J. Abello and J. Vitter, editors, *External Memory Algorithms, DIMACS Series, AMS*, 1999.

[2] J. Barr and L. Cabrera. AI gets a brain. *ACM Queue*, 4(4), 2006.

[3] Berkeley open infrastructure for network computing (BOINC). http://boinc.berkeley.edu.

[4] Boinc survey results. http://boinc.berkeley.edu/poll_results.php, as of July 27, 2006.

[5] distributedcomputing.info. http://www.distributedcomputing.info.

[6] distributed.net. http://www.distributed.net.

[7] The ESP game. http://www.espgame.org.

[8] The great internet mersenne prime search. http://www.mersenne.org.

[9] A. Holohan and A. Garg. Collaboration online: The example of distributed cmputing. *Journal of Computer-Mediated Communication*, 10(4), 2005.

[10] R. Hunicke, M. LeBlanc, and R. Zubek. MDA: A formal approach to game design and game research. In *Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence*. The AAAI Press, 2004.

[11] P. Hyman. 'casual' video games are serious business. *The Hollywood Reporter*, June 17, 2004.

[12] IDGA casual games SIG. 2006 casual games white paper, July 2006.

[13] A. Linn. Firms seek to cash in on 'casual games'. *USA Today*, July 9, 2006.

[14] Macrovision. Survey reveals casual gamers are not so casual. *Macrovision Press Release*, June 28, 2006.

[15] R. J. R. Mena. Using massively multi player technology for science education: P.A.S.T. history. In *Proceedings of International Conference on the Future of Game Design and Technology 2005*, October 13, 2005.

[16] Mudcraft. http://www.mudcraft.com.

[17] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Appl. Math.*, 120(1-3):197–207, 2002.

[18] P. Pardalos, J. Rappe, and M. Resende. An exact parallel algorithm for the maximum clique problem. In P. P. R. De Leone, A. Murl'i and G. Toraldo, editors, *High Performance Algorithms and Software in Nonlinear Optimization*, pages 279–300. Kluwer, 1998.

[19] Peekaboom. http://peekaboom.org.

[20] Phetch. http://peekaboom.org/phetch.

[21] L. F. G. Sarmenta. *Volunteer Computing*. PhD thesis, Massachusetts Institute of Technology, 2001.

[22] SETI@home. http://setiathome.berkeley.edu.

[23] L. Thimm, D. Kreher, and P. Merkey. A parallel implementation for the maximum clique problem. *Journal of Combinatorial Mathematics and Combinatorial Computing*. to appear.

[24] Amazon mechanical turk. http://www.mturk.com/mturk/welcome.

[25] Urban dead. http://urbandead.com.

[26] Verbosity. http://peekaboom.org/verbosity.

[27] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326, New York, NY, USA, 2004. ACM Press.

[28] L. von Ahn, S. Ginosar, M. Kedia, R. Liu, and M. Blum. Improving accessibility of the web with a computer game. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 79–82, New York, NY, USA, 2006. ACM Press.

[29] L. von Ahn, M. Kedia, and M. Blum. Verbosity: a game for collecting common-sense facts. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 75–78, New York, NY, USA, 2006. ACM Press.

[30] L. von Ahn, R. Liu, and M. Blum. Peekaboom: a game for locating objects in images. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 55–64, New York, NY, USA, 2006. ACM Press.

[31] Wikipedia. http://wikipedia.org.

[32] B. Winn and J. Tye. Crafting a web-based, nonviolent, real-time strategy game. In *Proceedings of International Conference on the Future of Game Design and Technology 2005*, October 13, 2005.