ERA: Environment, Rules, Agents

The Application of ERA in a Real-World, Tightly Constrained Problem

Graduate Teaching Assistants at the University of Nebraska Computer Science Department

Josh Reed

Abstract

Results of tests of an ERA-based algorithm when applied to a tightly constrained, or in some cases even over-constrained, problem. The focus is on how a time-efficient approach can provide a decently optimized solution (or partial solution) to the problem of Graduate Teaching Assistant assignments within the University of Nebraska, Lincoln Computer Science and Engineering Department. Because of the optimization aspect, a few changes had to be made from the original ERA Algorithm that was presented by Jiming Liu et al. [7] The results show that this approach is very good especially based on what can be statistically measured, showing that it is an effective way to find a good quality solution when there is not an infinite, or even large for that matter, availability of time.

The Application of ERA in a Real-World, Over-Constrained Problem

Graduate Teaching Assistants at the University of Nebraska Computer Science Department

The Problem

Real-world problems often present many challenges that do not present themselves as frequently in pre-defined common "trivia-based" problems. Following suit, the GTAAP (Graduate Teaching Assistant Assignment Project) is no exception. The goal is to automate the assignments of GTAs to courses, and thus eliminate some human error, as well as speed up the time it takes to figure out the assignments. In many cases, this particular problem is over-constrained and thus does not actually have a *solution*, in the sense that all the constraints are satisfied and all classes have GTAs assigned to them. So the task, then, is to be able to discover the solution that best satisfies as many of the constraints as possible, and additionally keeps the GTAs as happy as possible by respecting their preferences to as great an extent as possible. While this may seem a lofty task, using computational strategies to generate such a solution that can be the basis of such an assignment decision should be very possible.

The Process

In diagnosing the task and in trying to understand all the requirements of the problem itself, several avenues were explored. First, the decision to look at local search methods was made due to the fact that various systematic search methods were explored previously by others researching this problem, and the results were still lacking. The real trouble was with trying to limit the time it takes to run the algorithm. It is not practical for an algorithm to take days to find a solution, or even that long to find the best possible solution. While various means were taken to attempt to make those systematic search algorithms to run faster, it seemed appropriate to go a different direction, and use it for comparison.

So local search seemed the logical choice, especially since they are known for their speed. While they are expected to not always find the absolute best answer, they generally find a very good result and at the same time explore less of the search space, and thus reach their conclusion much quicker.

So, with this, the research began. It all started with looking into social choice algorithms. This seemed like a good approach since it relied on agents to keep track of their own preferences, and each would act so that it could improve its own "status," and thus the "happier" each individual agent is, then the better that particular solution is overall. This is quite an emerging topic, and so it was seemingly a perfect research direction. So many, many papers were read surrounding this idea. These included "Issues in Multiagent Resource Allocation" [2], "On Maximal classes of Utility Functions for Efficient one-to-

one Negotiation" [3], "A Rational Model of Cooperation Negotiation in Multi-Agent System" [6], "The 1st International Workshop on Computational Social Choice" [4], "Simulation of Negotiation Policies in Distributed Multiagent Resource Allocation" [1], "MDPOP: Faithful Distributed Implementation of Efficient Social Choice Problems" [8], and "Multiagent Resource Allocation in k-additive Domains: Preference Representation and Complexity"[5]. Unfortunately, after all this searching, the reading proved to not be as fruitful as anticipated. They all essentially stayed on a relatively high level, not delving as much into actual computational, algorithmic solutions. There was the discussion of how the problem was divided up into agents who were in charge of negotiating based on their own variables interests in gaining access to certain resources. They also all emphasized utility functions, which are essentially a way of determining the quality of a given solution both for the individual agent and for the problem as a whole. These utility functions can take a variety of forms: for the individual agents they could be based on the number of broken constraints, preferences of certain resources, or even a combination of the two. Then for the problem as a whole, it is always some form of a combination of the individual agent utility functions. Many times it appears as either the sum or the product of the individual agent utility functions.

These methods had the limitation of often being presented in a very high level, and not always having an obvious direct computational implementation. Additionally, most of the methods presented agents as strictly rational, meaning they will only give up their current resource(s) if the alternative is strictly better, based on its utility function. While this may not seem at first like a problem, it did not seem practical for our application because such a greedy, hill-climbing approach does not give much room for error. For instance, it would not be uncommon that with a particular assignment, an agent would to need to take a bit of a loss so that another agent could have an even greater gain, and thus improving the overall solution to the problem. Even in knowing that, these social choice methods presented did not make it possible for an easy alteration to those purposes. It would have been difficult to determine when such a situation exists since one agent is only responsible for its own resources and does not have the vision to foresee future moves made by other agents. With all these questions still unanswered after a reasonable amount of research, it is only logical that the shift be made into another direction of investigation.

Given the former direction, it seemed to have a lot of good elements, but just a few problems, so we wanted to avoid straying too far away from it. An option presented by Dr. Berthe Choueiry was to inquire into the ERA method, and attempt to apply it to this particular problem. Conveniently this method uses many vary similar ideas to social choice. It uses agents as the means of negotiation, and essentially its termination is determined by a utility function (minimize broken constraints was the utility function proposed in the paper introducing this method). Each agent acts individually and determines the utility of a particular move for its own purposes. The major place where ERA differs is that it introduces quite a bit more randomness. It also introduces a "better-move" strategy in which the agent does act in its own interest to move to a better position, but it does not necessarily have to be the best current position available (this will be explained further later). In this way it allows for a large variety of movement, and thus agents will have a less likelihood of just staying in one place, and causing the rest of

the agents to "suffer." So, ERA seems to have a lot of the advantages of local search while traditionally having the record of bringing pretty good quality results. Thus, it was decided to attempt to implement this method in the application.

The Particulars

It is now time to go into the details of this new implementation of the ERA algorithm. It should be noted that it is strongly based on ideas/algorithms designed by Jiming Liu, Han Jing, and Y.Y. Tang [7]. This particular implementation was coded in Java, and made use of an "Interactive Solver" system designed by Ryan Lim for use on this same problem (also in Java). The use of this code greatly reduced the coding overhead for this particular project, for instance by already handling the importing of the course, GTA, and constraint data from the database. It also already had several constraint types that were already constructed and managed their own checking functions. This meant that the majority of my task was to create the algorithm and fit it in with the current interface. While this is an advantage, there was still plenty of tasks that required attention.

The modeling of the problem sets up the variables as courses, with their domains being the GTAs. Each GTA has specified preferences for each course on the scale of 0-5, 5 being the highest preference, and this information will be used in optimizing the solution. There are several main constraints that are important to keep track of. First there is a mutex constraint that says that no TA can be scheduled for two different courses that meet at the same time. Next, there is an equality constraint that maintains that certain user-specified courses (or sections of the same course in many cases) must have the same TA. Additionally, there is a capacity constraint because each TA has a limited amount of time they are able to spend as a TA. Finally, there is also a confinement constraint, which are user-specified sets of courses where if a TA is selected for one of them, all other courses the TA is assigned to must be within that same group. With this modeling, the problem is prepared for placing a solution algorithm on top of it, in this case it is going to be a version ERA algorithm.

In the first phase, the ERA algorithm was implemented nearly strait based on the paper, which is based on the task on minimizing violations (also known as broken constraints). This method has three basic move types that an agent can perform, each respectively having a particular probability of occurring. These three types are random-move, better-move, and least-move. The first is very self explanatory, a random position is selected and moved to regardless of whether it is better or not than the current position. Better-move is a bit more complicated. It uses a random position, and then if it is better, it moves there, if it is not better, then it stays in its current position. Finally, least-move checks every position and figures out the absolute best position to move to, and then moves there. Like mentioned earlier, each of these would have their own proprietary probability of being selected as the move behavior.

This move behavior functionality was adapted a bit, based on the paper's subsequent mentioning of alternative movement selection behaviors that are still based on these

movement principles. Liu et al mentions a strategy called rBLR, which basically has a probability for a random move occurring, and then the only other option if a random move does not occur would be to do a better-move "r" number of times and if still unsuccessful a least-move would occur. So basically if better-move fails to produce a better move, then better-move will be repeated "r" times, and finally if it fails all "r" times the least-move would occur. This was the approach selected, partially based on research by Zou Hui [9]. Another result of Zou Hui's research was the use of 2% as the random-move probability, and thus the remaining 98% of the time rBLR would be selected. The value for "r" used in this instance was 3, which seemed to give good results, and going higher did not seem to provide any better or faster results. Again this was a discovery based on experimental attempts.

Next, after the implantation of this phase was completed, the addition of slightly better optimization was a must. It seemed best to make as a secondary level of determining a solution's quality based on the preference ranking that the assigned TA has for the particular class he/she is assigned to. The overall problem's overall utility function (in respect to the TA preferences) is defined as follows:

$$\sum_{agent \in Agents} \textit{agent's assigned TA's preference for this class}$$

The method of performing the sum of all the agent's preferences was chosen mostly out of experimental trials. First the *product* of all the preferences of the assigned TA's was computed which gave a lot of weight to the "0" preference, making the entire utility zero in such a case that even only one "0" preference exists. This at first seemed to be an advantage, because it would be best to avoid all zeros if possible. But because the problem is so tightly constrained and usually does not possess an absolutely perfect solution, the results in many, many cases had to have a zero in at least one TA's preference, making the utility function essentially useless. So the sum was computed instead so that at least some differentiation between solutions could be evaluated. This unfortunately loses a lot of the weight of a "0" preference that was desirable, but sometimes compromises must be made, especially in dealing with local search methods.

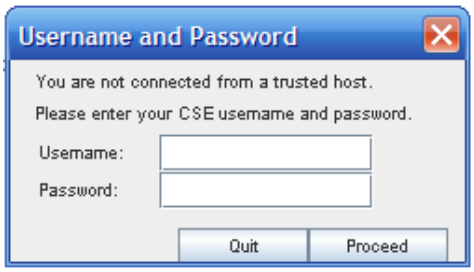
This optimization was added in as a secondary form of optimization. First a better solution would be determined by the number of constraint violations, so the storage of the best solution would be replaced with this new, lower number of violations solution. The only other way that the storage of the best solution would be replaced is if there were an equal number of constraint violations, but a greater overall preference utility. This provides that at the same level of constraint violations, the best solution is the one that has the maximum utility.

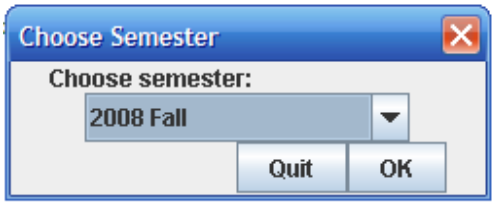
It should also be noted that each agent now determines a better move as a position that has strictly less violations or as a position that has equal number of violations and a higher preference value. Surprisingly, this addition itself proved to even benefit the quicker finding of lower violation solutions, making it much easier to find a zero constraint violation solution. While this was found in practice, it does make some sense

on the theoretical level as well. If more movement occurs, solutions should be found quicker.

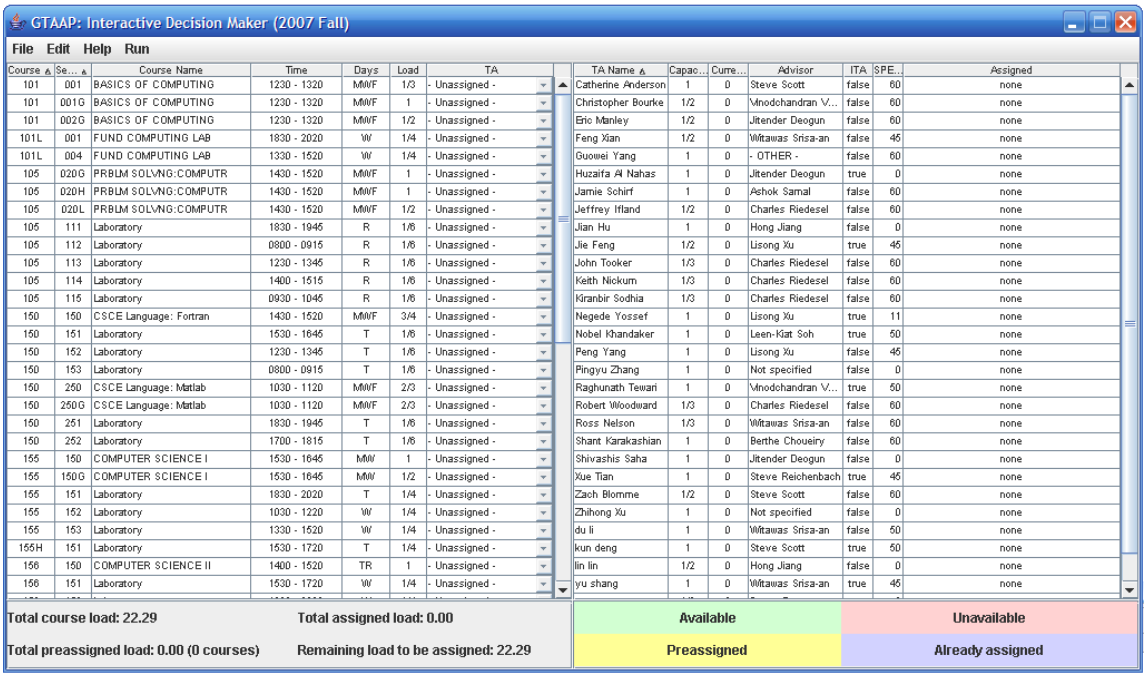
The Program

The program begins with either a login box or goes directly to the semester selection box (pictured below). This is based on if the connection is being made from cse.unl.edu or from the outside. If it is from the outside, then a CSE username and password must be supplied to set up an SSH Tunnel for the database communication. After a successful login, one can select the semester for which they wish to assign GTAs.

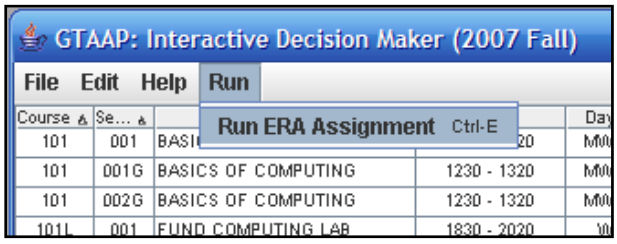




After the semester has been selected, then the semester will be loaded from the database, which may take a minute depending on your internet connection speed. Then the main body of the program will become visible (pictured below). This screen is set up with the left side being the "task pane" and the right side being the "resource pane." Essentially the left side shows the variables (courses) and what values (GTAs) are assigned to them. On the right side is the list of resources (GTAs) and what variables (courses) they are assigned to.



From this view, one can manually assign TAs and then Arc-Consistency is performed to denote how the change affects other variables. Doing such a task manually might seem a little overwhelming, which is why ERA has now been implemented in this interactive solver. To run ERA to get at least a starting point or perhaps a perfectly acceptable solution, just go to the menu and select Run → Run ERA Assignment. This will bring up a box in which "OK" can be selected to begin the ERA assignment process.



The algorithm terminates only upon the completion of 150 time steps. Time steps are incremented essentially once each agent has had an opportunity to make a move. So once all the agents in the system have made their moves (or chosen to stay put), the time step is incremented. The number of time steps is the sole termination condition because with the optimization implemented, there is no practical way to determine when it has been "fully optimized," so it must be cut off at some point. That point selected was 150 time steps. This number of time steps seemed to be a fairly a fairly decent compromise between time and quality. It appeared, based on numerous test runs of the algorithm, that very little improvements were made after 150 time steps, and in actual runtime 150 time steps seemed to not be a too tortuous amount of time to wait. It should be noted though that each time step may take longer for problems with a larger number of variables and/or larger domain sizes.

The interface still needs a little bit of work, and thus still has a few quirks to it, but overall it is still very usable. Once such quirk is, for instance, when the progress bar reaches 100%, the ERA assignment is completed, but instead of automatically progressing to the next screen at that point, you have to hit the "Done" button to get to the results screen. This screen is where it reports if there were still any violations in the best solution or not, and warns the user that these will be marked "Unassigned." Click "Done" again to now close the window.

Then the main window should be automatically updated with the results of the ERA assignment. In the bottom-left gray box is the "Current Quality" of the solution. This is basically a percentage and is based on the sum of the assigned GTAs' preferences divided by the total possible preference score which would be 5's for every course (which is nearly impossible). Anything above a 0.50 would denote that most of the GTAs are either happy about or neutral to their assignments.

The Results

There were several semesters of Data that were available for testing and comparison to the "actual" solution that was recorded in the web interface of GTAAP. The results for the Fall 2007 semester provide interesting insights:

Fall 2007 *

	ERA Algorithm	Actual Assignment
Quality/Utility Percentage	0.68	0.66
Course Loads Remaining Unassigned	0.00	3.48

These results show that the ERA Algorithm has a slight advantage over the human-generated solution in overall numbers. This is incredible considering the ERA Algorithm took only a matter of minutes to run! The human-generated solution is guaranteed to require much, much more time to produce. Granted, the human-generated solution can take into account some intangible factors that a computer cannot in determining who would make a good TA for a particular course. Since those intangibles are somewhat necessary to take into account, a human will be making the final decision and taking into account the details a computer cannot really know about. In any case, it seems this computational result will likely improve the end result of the assignment.

Spring 2007 *

	ERA Algorithm	Actual Assignment
Quality/Utility Percentage	0.65	0.62
Course Loads Remaining Unassigned	0.33	4.16

While the ERA algorithm did not get down to zero unassigned courses for Spring 2007, it did get pretty close, with only one class remaining unassigned. This result is a bit deceiving though because there were a couple of "0" preferences in that result. One trend that was a little bit noticeable in the Fall 2007 data and very obvious in this Spring 2007 data was that in leaving more courses unassigned, the human-generated solution was able to better maximize the individual TA preferences. This is how it is possible for there to be several fewer assigned classes but still a relatively close Quality Percentage.

Fall 2004 *

	ERA Algorithm	Actual Assignment
Quality/Utility Percentage	0.57	0.58
Course Loads Remaining Unassigned	0.75	2.00

In this dataset (Fall 2004), though a bit of a different result appears. The Qualities are essentially the same, while the assigned course load is slightly better for the ERA algorithm. Thus these two are very comparable statistically, but the human-generated

* These results are not set in stone, each time the algorithm is run, different results of both higher and lower quality are possible due to the randomness factor of the algorithm.

actual assignment has the advantage of knowing which classes are best to leave unassigned, and which ones are absolutely necessary to have a TA assigned to it. In any case though, again, the ERA Algorithm's result would make a good starting point for tweaking by a human user.

These datasets were chosen partially because they are fairly representative of various situations in the data: GTA load exceeds (by a little bit) the total load to be assigned (Fall 2007), GTA load and total load to be assigned are very close to the same (Spring 2007), and GTA load is a little less than the total load to be assigned (Fall 2004). Each of these semester comparisons of actual assignments to the ERA results was very tedious and thus some datasets that would require additional setting up were left out. Also the datasets for the current two semesters, Spring 2008 and Fall 2008, were left alone because those are both still currently being used, which would make changing things in the data, in order to do such a comparison test, dangerous and potentially harmful to the system.

Limitations and Difficulties

There were several difficulties that were experienced that caused setbacks in the research. One of these difficulties was the fact that this is a live system, meaning when it is having problems or something needs to be updated or changed I often had to set my research aside to fix the pressing problems. This mostly meant that I was not able to get around to putting more effort into improving the user-interface and fixing a few of the quirks (many of which I inherited with this code).

Along similar lines, the code that I inherited in some ways was not very well structured, and so many rabbit-trails had to be searched in order to understand how certain things are working.

Additionally, another limitation in the testing of the results is that the final assignment data I was getting from within the GTAAP system. This is a limitation because during some transcribing of some data there were several indications that maybe some parts of the data may have been out-of-date based on section additions or multiple TAs assigned to the same course as the "final assignment," without that course being duplicated in the database as is the standard procedure in such a case. But overall, the representative data should be at least an approximate representation of the actual assignment and of human-generated solutions in general. An additional unfortunate aspect of the data is that since in recent years the assignment of GTAs has been done by hand by a human, they relied on his knowledge of their preferences and skills rather than accurately expressing their preferences in the system. Due to this there are a particularly high quantity of "3" preference ratings, which is the default, essentially a neutral-to-the-course value. The system does not know that this is the case though, and assumes that all these neutral preferences are legitimate. Thus, this makes the statistical representation of both the actual assignments and the ERA algorithm assignments lose accuracy in comparison to the real-world value of a particular set of assignments.

Another limitation, which has been mentioned before, is that human-generated solutions have the advantage of personal contact and other intangibles that are hard to represent to in a computer. By personally knowing the TAs, a human can easily determine who would make a good TA for a particular course, while a computer does not have that advantage. So as further tests are continued in the real-world, it will be interesting to see how well these intangibles are handled by the computer-generated results.

Conclusion

Overall this project has accomplished its given task: to use Constraint Processing to aid human users in making decisions concerning the assignments of the Computer Science and Engineering GTAs. ERA claims to be very quick, and I would confirm that to be absolutely true. ERA also claims to get good results in this short amount of time, and I believe that as the results have shown, it really does get very good results, especially statistically (which makes sense since it runs based off of such statistics).

Also it has been proven that ERA can be used in over-constrained problems to find a very good partial solution. Such a situation would cause a systematic search model some problems, if not cause it to fail completely. In conjunction with human verification, which would generally be needed anyway, ERA performs perfectly to ease the pains of the alternative—a manual assignment process.

This particular instance that was based on the ERA algorithm also used some additions, which really proved to better the search results. The main addition was a form of optimization based on preferences. This was used as a secondary form of verifying a better overall solution. It proved to be very, very important in practice, not only in maintaining a higher quality of solutions, but also in forcing more movement to truly better positions by agents, allowing better solutions (based on constraint violations) to be found quicker.

Overall this is a very recommended algorithm basis for some types of problems. In general, it should perform well on real-world problems as well as problems were time is key and decent solutions are needed in a short amount of time.

Future Research

There are still a vast many things that need to be investigated in the future. First and foremost, once the upcoming fall semester is ready to go and has all the GTAs “hired” within the system, then it would be great to test how much it really helps the human-user in producing a solution for the assignment. This would test more for the intangibles rather than pure statistics.

Another way to further gain better results would be to have a mechanism for a user to specify which courses are of low priority and thus can remain unassigned. This will most likely allow for more GTAs to get assigned a course that is higher on their preference list. This would also make the solution a little bit more similar to the human-generated solution, and perhaps giving it more value in the eyes of the user.

More future research could be in other local search methods, such as genetic algorithms, among others, to see if such an approach would yield as good or better results than this ERA-based local search method. Additionally, if other local search methods are implemented, it might be possible that a hybrid model of two or more methods might provide better results. For instance if two algorithms are being run on two different threads they may be able to pass their best solutions so far back-and-forth, which might perhaps yield a higher quality result. Also it might be interesting to investigate if running several time steps of the ERA algorithm, and then passing the result(s) onto a genetic algorithm to start out with (so that it would have a better beginning generation), would result in higher quality, or perhaps faster, results.

Also, taking into account that the assignment is currently human-generated, it may be interesting to take an approach that is somewhat human-like. Humans generally take a "greedy algorithm" approach in which they may make a few back changes, but overall they move forward along a set prioritization scheme. It might be interesting to interview the current person in charge of the GTA assignment and determine his process in making assignments, and then attempt to produce a computational model that is similar to his approach. Since it would be in essence a greedy algorithm that still can change a bit of the past, it would certainly be very, very quick in the runtime. Additionally it might give more human-like results, which may be a big advantage for some users. Finally, it would also be an attempt towards one goal of Artificial Intelligence: making computers behave more like humans. While there is never a desire to replace humans, it can be very important for computers to take away work that humans traditionally performed but would rather not put the computational energy into. It is a lofty goal, but sometimes goals need to be set high so that true achievement can really become possible.

References

[1] Buisman, Hylke, Gijs Kruitbosch, Nadya Peek, and Ulle Endriss. Simulation of Negotiation Policies in Distributed Multiagent Resource Allocation. In *Engineering Societies in the Agents World VIII*, Springer-Verlag, 2008. Postproceedings of ESAW-2007. To appear.

[2] Chevaleyre, Yann et al. Issues in Multiagent Resource Allocation (2005) 3-31.

[3] Chevaleyre, Yann, Ulle Endriss, and Nicolas Maudet. On Maximal Classes of Utility Functions for Efficient one-to-one Negotiation. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*, pages 941-946, August 2005.

[4] Endriss, Ulle. The 1st International Workshop on Computational Social Choice (2007).

[5] Chevaleyre, Yann, Ulle Endriss, Sylvia Estivie, Nicolas Maudet. Multiagent Resource Allocation in k-additive Domains: Preference Representation and Complexity (2004).

[6] Li, Bang-Quing, Jian-Chao Zeng, Meng Wang. In *Proceedings of the First International conference on Machine Learning and Cybernetics, Beijing, 4-5 November* 2002, pages 1332-1335. A Rational Model of Cooperation Negotiation in Multi-agent System (2002).

[7] Liu, Jiming, han Jing, Y.Y. Tang. Multi-agent oriented constraint satisfaction (2001).

[8] Petcu, Adrian, Boi Faltings, David C. Parkes. MDPOP: Faithful Distributed Implementation of Efficient Social Choice Problems (2006). ACM.

[9] Zou Hui and Berthe Y. Choueiry. In *Workshop on Applications of Constraint Programming* pages 81-101. Characterizing the Behavior of a Multi-Agent Search by Using it to Solve a Tight, Real-World Resource Allocation Problem (2003). Kinsale, County Cork, Ireland.