Mechanismic Task Allocation: A Comparison of Utility and Variance Across Designs

By: Mark Grozen-Smith, Ernesto Gaxha, Buffalo Hird

Abstract:

From families to large corporations, groups of people often struggle to solve the problem of who does undesirable tasks. While many mechanisms exist for optimizing task assignment when money is at stake, we explore methods when money is impractical or distasteful to use. The methods are a modified VCG, Random Serial Dictatorship (RSD), Top-Trading Cycle (TTC). Additionally, attention is given to determining allocation and payment via Sperner's Lemma. We assess each mechanism using a well-defined notion of fairness that primarily maximizes overall utility. The modified VCG proved to be the best algorithm for minimizing overall agent disutility, though rather volatile between agents' utilities per iteration. RSD and TTC performed very similarly on our data, both with less volatility but also less efficiency in the task allocation. The modified VCG achieved the best task assignment that minimized the risk of agents receiving their least desired task, thus we recommend using this when trying to do best for the set of agents as a whole over time, though when trying to just make the allocation as even as possible, TTC or RSD would be best.

I **Introduction:** (motivation, what are you doing, summary of results, related work)

Many of the biggest challenges we encounter in our daily lives come down to the problem of task assignment. Often, a group of people must accomplish a set of tasks, but each person has a self-interested allocation of the tasks that is likely different from those of the others. This naturally raises the question of how to fairly divide the tasks among the group. Often what complicates this assignment issue is the lack of money to organize assignment. In other instances, assignment is made simple with monetary calculations and transfers of utility. Employees are rewarded with salaries for the completion of tasks required by employers. The delegation of work is based largely on a combination of skill and preference, which is at least partially reflected in the price.

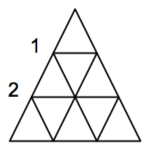
In some assignment problems, however, the use of money is either impractical or distasteful. Examples of such a problem are numerous. Consider household chores; a family or group of roommates could not reasonably use money to organize who does what, since money-service exchange between friends and family is frowned upon. Moreover, such a solution would favor those with more money. If one roommate is particularly wealthy, (s)he could simply pay her or his friends to do all the work.

We plan to address this issue by applying several different assignment mechanisms to generated assignment problems. Our mechanisms include VCG, Random Serial Dictatorship, division based on Sperner's Lemma, and Top-Trading Cycle Mechanism applied to the House Allocation problem. We often must modify the mechanism to better suit our needs, as explained in the methods section (for example, we modify the VCG payment).

Background

The literature in this area is relatively new because, while there is a wealth of research done on allocating things people do want, there has been less work done studying how to allocate things people do not want (i.e. chores). While Random Serial Dictatorship and Top-Trading Cycle were more directly applicable to this area, we note that many of the desirable properties of VCG fail to hold when modifying the mechanism to include budgets and bidding against performing tasks instead of for resources. We researched modifications of this mechanism such that we could best retain VCG's benefits. We considered an implementation which reverses the payments such that a winning bidder is the one who bids the smallest magnitude negative value, such that they are least averse to performing the task. We then redefine externality such that externality imposed by the winning bidder to the runner-up bidder is paid by the winning bidder to all others. As a result of negative values, this means that given a runner-up bid of x, each non-winning bidder must pay the winning bidder externality -x/(n-1), where n is the total number of bidders. We believe this to be a suitable mechanism as given this, a user cannot escape payment by misreporting and doing so might only 1) if they bid higher increase how much they might implicitly pay as they might now have to do a chore they strongly negatively value 2) if they bid lower reduce their chance of doing a chore they more prefer while instead having to pay to not perform it.

We also utilized Sperner's Lemma and the related PPAD-hard graph algorithm. We first discovered Sperner's Lemma by chance described in a New York Times article - cited following the conclusion (Sun, Albert) - on dividing housing costs among roommates. It uses a cake-cutting (fair division) algorithm which is concerned with best subdividing a set of n resources among n agents such that there is an envy-free allocation. This algorithm was first implemented by Forest Simmons and has a guaranteed solution with very few constraints. Sperner's Lemma scales to any n-dimensions of simplexes - triangles extended into n dimensions - but we define it in terms of 2-dimensions with 3 agents considering 3 resources. We create a triangle T which is triangulated into smaller number of isomorphic elementary triangles defined by k. We show T where k = 2.



We proceed to label the vertices of this simplex, defining only two simple constraints to guarantee an envy-free allocation. We label the vertices such that, given a value assignment or cost defined at this vertex, a given agent would choose a given resource. We define a total amount to be paid C such that values are assigned with each corner of T corresponding to a given resource having cost C and all other resources having cost 0. For other vertices, C is subdivided such that each resource has higher cost linearly proportional to distance to the given inner vertex. We require that (1) the corner vertices of T have different labels such that each resource is desired once and (2) the labels along the edges of T match one of the two labels of the corner vertices which span these outer edges. Given these two constraints, we can proceed to assign any label to the interior vertices. This results in Sperner's Lemma:

> "Sperner's Lemma for Triangles. Any Sperner-labelled triangulation of T must contain an odd number of elementary triangles possessing all labels. In particular, there is at least one." (Su, Francis E.)

This means that given any labelling, we will have at least one triangle which includes each agent desiring a unique resource. As a result, determining this triangle determines a one-to-one, envy-free allocation of n resources to n agents. Given these allocations, we can average the vertices to get an assignment closer still to the ideal equilibria. Alternatively, we can redefine T to have a higher k value such that we have a more precise set of inner triangles. We leave the proof of Sperner's Lemma as a note in the works cited, but essentially it is a variation of the pigeonhole principle by which a satisfying allocation is necessary. This algorithm scales nicely for n-dimensional simplexes though we focus on the 2-dimensional triangle implementation as it is more intuitive and relevant to our model. Perhaps more importantly, it is more feasible to reason about this PPAD-hard algorithm with small n and k values.

II Model:

We can see that before we truly investigate our solution, we must make some statement about what it is we want out of our mechanism. Of course, if we simply wanted task assignment, we could simply assign tasks randomly, but we are in search of an idea of fairness. Here, we define what we expect out of a fair algorithm:

- Tasks must generally go to those most willing to perform them. (In other words, it must attempt to generally maximize utility or minimize negative utility).
- The mechanism, ideally, must be strategy-proof and truthful.
- The mechanism, ideally, will allow for multiple combinations of task assignments, and not just one agent-one task.

When running our simulations, we will refer back to this to evaluate our mechanisms.

In our simulations and examples, we will use the same few sets of inputs to make comparisons easier, then also compare statistics across algorithm performance on a large sample of test runs. The first examples we run through are simple. Each agent has a preference order, determined by bids, and all agents start with an equal budget. A budget is not actual money, but points that can only be used in further iterations of the mechanism. We will call this Agent Set 1:

Agent 1: {1,2,3,4}	Bids (1):{-1,-5,-9,-13}	Task 1: {1,2,3,4}
Agent 2: {4,3,2,1}	Bids (2):{-2,-6,-10,-14}	Task 2: {3,2,4,1}
Agent 3: {4,1,2,3}	Bids (3):{-3,-7,-11,-15}	Task 3: {3,2,1,4}
Agent 4: {3,2,1,4}	Bids (4):{-4,-8,-12,-16}	Task 4: {4,3,2,1}

Rather than one explicit set, Agent Set 2 is a summary of a large amount of test data so that we can make statements about the expected performance of the algorithms we are analyzing here. The test data sets are created by forming lists of between 2 and 20 agents and tasks, giving tasks a random preference ordering of agents to do those task (this is necessary in the TTC algorithm), then choosing bids from the agents semi-randomly for each task. The bids could be chosen in several different ways, dependent on the focus of the study. We decided to keep the bids to be a simple ranking from -10 to -1 on how much an agent does not want to do a certain task. Other studies may prioritize other factors such as the ability for an agent to ensure against doing a task by throwing an entire budget at that bid, but these factors are subjective to the priorities of the study and the exact agents in the real-world use case. We then created 100 test sets of this data and found the average total utility of each algorithm run on these 100 test sets as well as the average variance across the utilities of the agents in these test sets.

III **Methods**:

Our modified VCG works as follows: There is an iteration for each task. In one iteration, it takes in as input the task being auctioned, and each agent's bid. It assigns the task to the agent with the highest bid (in this case, the least cost of doing the task), but the agent who wins gets paid the amount of cost avoided by the agent who would have won had the winner not been there. This payment to the winner of the auction then increases his/her budget for the current or future set of task allocations. This payment is split evenly among the non-assigned agents, and comes from their budgets. Then this is repeated for the next task, until all tasks are assigned. Applying this to Agent Set 1, we see that:

Agent 1 wins Tasks 1 and 2, by bidding -1 and -5
Agent 2 wins Task 4 by bidding -2
Agent 3 wins nothing
Agent 4 wins Task 3 by bidding -4

To calculate utility, note that in each case, the winning agent gets paid by the others. For example, Agent 1 wins task 1, and receives negative of the disutility of -7 avoided by Agent 3, a payment of 7 units, split evenly among the three others, or a per agent cost of 2.33. However, since payment to Agent 1 equals total costs to the others, we can ignore this when finding total utility, and just sum the agent's bids for each item. This becomes (-1) + (-5) + (-2) + (-4) = -12.

Our Random Serial Dictatorship mechanism works as follows: As input, the mechanism takes in each agent's preferences (determined by the bids). Then, in each iteration, the first remaining agent is given his or her top choice, then removed from the pool of unassigned agents. This continues until all agents have been assigned a task. Applying this idea to Agent Set 1, we see that:

Agent 1 wins task 1
Agent 2 wins taks 4
Agent 3 wins task 2
Agent 4 wins task 3

To calculate utility, we use the standard notion of value minus payment. Here, however, we know that value of doing a task is represented by the bids, and since there is no payment associated with the Random Serial Dictatorship mechanism, we see that utility equals value. Thus, total utility equals (-1) + (-2) + (-11) + (-4) = -18.

Our TTC mechanism works as follows: As input, it takes in each agent's preferences (determined by the bids) and each task's preferences (determined randomly or by some method external to the TTC mechanism). At a high level, TTC works by having both agents and tasks point to their desired match, finding agent-task cycles, and removing them from the set still in need of matching. The agent-task cycles are parsed to give agents their task they prefer, this was decided to maximize satisfaction by the agents. This is repeated until there are no more agents in need of matching. Applying this idea to our Agent Set 1, we can see that Agent 1 and Task 1 point to each other, creating a cycle. This is removed. Then we can see that there is a cycle where Agent 3 -> Task 4 -> Agent 4 -> Task 3 -> Agent 3. Thus, this is removed. This leaves Agent 2 and task 2, which match to each other. We get the matching:

Agent 1 wins task 1
Agent 2 wins taks 2
Agent 3 wins task 4
Agent 4 wins task 3

As in the RSD mechanism, the value of doing a task is represented by the bids, and since there is no cost associated with the TTC mechanism, we see that utility equals value. Thus, total utility equals (-1) + (-10) + (-3) + (-4) = -18.

Our Sperner's Lemma mechanism works as follows: As input we supply a desired n and k, where n is currently set always to 3. We work with k=1 or k=2 as these cost intervals are small and possible to easily reason about. We then provide defined preferences or randomly generate them for each agent at each of the n*k vertices in the graph. We then generate this graph and then walk through it to determine the top choice of a given agent at each vertices semi-randomly (per Sperner's). We then upon finding a satisfying allocation visit neighboring vertices to find the inner triangle which contains all labels. We then take the midpoint, averaging the values at each of the triangle's vertices to get the final values that each agent will be charged for each resource.

## IV	<u>**Results and Discussion**</u>**:**

The following chart displays results for both Agent Sets under each mechanism, for total utility and for average variance of each agent's utility.

<u>**Total Utility and Variance for Agents Under Different Mechanisms**</u>

Mechanism	Set 1 Utility	Set 2 Utility	Set 1 Variance	Set 2 Variance
Modified VCG	-12	-15.29	38.6667	15.68
Rand. SD	-18	-25.11	20.3333	3.96
TTC	-18	-25.88	15.000	3.92

An interesting result of the VCG mechanism is that there is a compensation for the people doing the tasks from the people not doing tasks. We can see in the simple example of Agent Set 1, that Agent

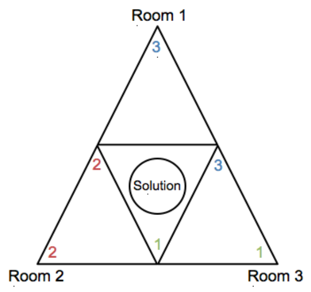
1 does two tasks. Thus, she receives payment from the other agents, which she can save for the next iteration of chores, while other agents, such as Agent 3, bid higher, get assigned less tasks, and drain their budgets. In other words, our modified vcg allows agents to optimize leisure over time more efficiently, by effectively borrowing or lending leisure with others. Whereas in the RSD and TTC, each agent had no choice but to do at least one task in each iteration. This also explains why there is a higher variance for VCG, since for each iteration, for an individual agent, utility could range much more widely. This also explains why there is a generally higher average utility, since we allow those willing to take on more tasks to take on more tasks, and give more leisure to those who would receive a large disutility from work per iteration.

This brings us to the issue of equity. If we wanted to reach a level of fairness over each iteration, one could argue that our modified VCG is not very fair, since the variance of utility for each agent is high. If we wanted a system in which every agent generally received the same level of utility consistently, then the TTC or RSD would be better, but doing this would not even be in the interests of the agents, if they wish to optimize utility over time.

One thing to note about TTC and RSD in the data above is that they seem to mirror each other pretty closely. The results from these two algorithms is very similar, but we believe this is to be expected. With the input data, the tasks have completely random preference orders on agents and thus the TTC algorithm becomes less enlightening. Normally, a task's preference on agents should reveal information about who is competent and this should be a trend among several tasks. Given that the data is just random, however, the TTC algorithm becomes only slightly more enlightening than RSD. This slight improvement is hard to see though since we have no information on agent competence in these generated data sets. Empirical data would be very useful in actually displaying the difference here.

Sperner's Lemma

We run this algorithm on the tractable parameters for which it can be reasoned about. We consider 3 agents with 3 resources and total budget of $1500 such that their combined spending cannot exceed or fall below this threshold. We run with a set of randomized preferences for each of 3*2 vertices and receive the following output allocating: agent #1 to resource #3, agent #2 to resource #2, agent #3 to resource #1. As this answer is in the center of the center triangle, we have an even split where each agent pays $500 or a third of the total budget. We expect to have to run this algorithm with exponentially higher k-values to achieve significant improvements in accuracy, but note the algorithms promising properties. Due to the independence of each vertex a certain level of truthfulness and robustness is expected of the algorithm as described in its original publishing. We attach an annotated solution graph here.



V Limitations and Further Work:

We outline some limitations in our approach and how further work could either expand upon or further address our current limitations in design and scope. Our implementation of Sperner's Lemma uses a carefully defined set of circumstances such that it is tractable and easily reasoned about. In addition, due to the need to iterate through an n-dimensional list, it is impractical to produce easily-understood code which accomplishes this while supporting any reasonable n. Furthermore, to produce results from this algorithm requires supplying a (n+1)-dimensional list of test data which models agents' preferences for each vertex. Due to the random walk of this algorithm, it is necessary to have all preferences defined as any agent's might be used at any given vertex. As a result, our analysis of Sperner's Lemma is confined to 2-dimensional triangles representing 3 agents and 3 resources. In further work we would further generalize this mechanism, which would entail utilizing far more cryptic, black-box like code and test data, which would be suitable for further analysis given that the mechanism itself satisfies our definitions.

In terms of the data we used to determine our results, several factors could be difficult to determine. In creating the data, we randomly determine the bid each agent places on each task, however, realistic situations could differ. For instance, it would make sense to have more consistent evaluations of certain tasks (i.e. everyone is going to bid in a way that says they would rather take out the trash than clean the entire garage), however, there are so many factors that make this unpredictable and subjective, this can only effectively be resolved by using truly realistic, empirical data.

Our discussion of differences in equity between our modified VCG and the TTC and RSD mechanisms hinges on the fact that they differ in the ability of agents to take on more than one task. In order to determine the difference between the TTC and VCG mechanisms themselves, we could have it such that our agents must be assigned to one and only one task, but this was out of the scope of this project.

VI **Conclusion**:

Our results show that our modified VCG delivered the highest utility and allowed for optimization over time. Looking back to our predetermined guidelines for our desired mechanism, we see that all our mechanisms, in one way or another, gave tasks to those more willing to perform them, but we can see that our modified VCG did this better on average. Also, our modified VCG allows for multiple combinations of task assignment, which may be what gives it the advantage in maximizing total utility, however this may be logical given our results as the modified VCG approach is best when trying to optimize for overall utility, not instantaneous fairness. When aiming for fairness within an iteration, TTC or RSD would be more ideal. In terms of truthfulness, we can see that in our modified VCG, agents have no incentive to misreport their preferences. Similar to the original VCG, this mechanism has agent independent pricing, and is agent-optimizing, because fixing the reports of the others, each agent’s payment is independant of her report, and the mechanism chooses the available alternative that maximizes that agents utility.

We note that Random Serial Dictatorship inherently embodies many of these properties due to its randomized nature. It is strategy-proof and has agent independent pricing as agents can do nothing to affect the random order in which tasks are chosen, having the option instead to only attempt to obtain the best remaining option still available. An agent’s preferences are unaffected by those of others inherently and therefore this mechanism also has agent independent pricing and is agent-optimizing as an agent will logically pick their preferred option given a set of options.We note that this holds as well for TTC, as stated in the text.

Works Cited:

Parkes, David C., and Sven Seuken. “ECONOMICS AND COMPUTATION: A DESIGN APPROACH.” *N.p.: n.p., n.d.* Web. May 2014. DRAFT

Su, Francis E. "RENTAL HARMONY: SPERNER'S LEMMA IN FAIR DIVISION." *American Math Monthly* 106 (1999): 930-42. Web. 5 May 2014. <http://www.math.hmc.edu/~su/papers.dir/rent.pdf>.

Sun, Albert. "TO DIVIDE THE RENT, START WITH A TRIANGLE." *The New York Times*, 28 Apr. 2014. Web. <http://www.nytimes.com/2014/04/29/science/to-divide-the-rent-start-with-a-triangle.html>.