



**PROJECT: COMRADE – Combinatorial Reverse Auctions: Design of
Efficient Mechanisms for Crowdsourcing Markets**

Funded under the 1st Call for HFRI Research Projects to Support Faculty Members and
Researchers and Procure High-Value Research Equipment

WP6: Design and analysis of mechanisms for combinatorial reverse
auctions

***Deliverable D.6.1: Technical report on data acquisition and implementation aspects of
the evaluation platform***

Prepared by the team members:

**Michail Fasoulakis, Evangelos Markakis, Thanassis Papaioannou, Georgios Papasotiropoulos,
Panagiotis Tsamopoulos**



ΕΛΙΔΕΚ.
Ελληνικό Ίδρυμα Έρευνας & Καινοτομίας

1 Introduction

This deliverable concerns WP6, and in particular, it is the first deliverable of WP6. It was completed by Month 31, in accordance to the amended timetable of COMRADE. The goal of the deliverable is to provide an overview of all the preparatory steps that were taken so as to be able to perform an experimental evaluation of selected algorithms and mechanisms. These steps concern the decisions that were taken (i) for creating synthetic data sets that resemble real world instances, (ii) for exploring the possibility of acquiring actual data from the industry, and (iii) for implementing the selected algorithms. The results of the experimental evaluation that took place after all these steps were completed, are described in the second deliverable of WP6 (Deliverable D.6.2).

We briefly outline the structure of this deliverable. The project COMRADE evolves around reverse (procurement) auctions, where a set of tasks needs to be assigned to a set of interested agents. The two major algorithmic problems for which we wish to perform an experimental evaluation are described in Section 2, and have also been described in previous reports, related to WP4. The former of these problems is a social welfare maximization problem under a given submodular valuation function for the auctioneer, whereas the latter one is a cost minimization problem arising in the context of spatial crowdsourcing. In Section 3, we describe the process we followed for data derivation and acquisition regarding the problem of social welfare maximization. In analogy to this, in Section 4, we do the same for the auction models that concern spatial crowdsourcing markets. Finally, in Section 5, we comment on implementation aspects for both auction models.

2 Basic Mathematical Models and Algorithmic Objectives

For the sake of completeness, we recall the general setup and the algorithmic questions we have been interested in. These have also been described in the deliverable D.4.1, which was the first deliverable of WP4.

We consider a procurement auction setting, and let $A = [n] = \{1, 2, \dots, n\}$ denote a set of n agents who are willing to participate in the reverse auction. The auctioneer is interested in having some predefined set of tasks completed, possibly subject to certain constraints, and is therefore aiming at hiring some of the participating agents. There are various possibilities for modeling the capabilities of each agent, the requirements of the auctioneer, and the bidding interface so that the agents can make offers on the tasks they can complete. E.g., the auctioneer may simply want to assign each task to a single agent, or assign tasks multiple times (for quality assurance). Each agent may be capable of performing a single task out of the tasks posted by the auctioneer, or she may be able to bid for performing a subset of the tasks. The interface may also allow agents to make multiple bids on different combinations of tasks, or to make bids only for single tasks. The choices made for these aspects can lead to a variety of different models for task assignment.

Based on the current state of the art, and on our Deliverable D.2.1 from WP2, which performed a literature survey, we decided to work on the following types of optimization problems that capture different aspects of task assignment in reverse auctions. We describe them separately below.

2.1 Social welfare maximization

We consider a generalization of a class of models that were originally introduced by Singer [3]. In these problems, the auctioneer asks each agent i to submit one parameter, denoted by c_i to the mechanism, corresponding to the cost for executing the tasks she is capable of. This assumption can capture several different scenarios. E.g., for an agent i , her cost c_i could simply be the cost incurred to her for executing a single task, among the tasks advertised by the auctioneer. In other cases, it can denote the cost for executing a collection of tasks that she is capable of doing (and in which case the agent should declare along with c_i the list of these tasks). Regardless of the interpretation of the cost c_i , all these settings can be treated in a unified manner.

Furthermore, to evaluate the selected outcome, the auctioneer is also associated with a valuation function $v : P(A) \rightarrow \mathbb{R}$, where $P(A)$ is the powerset of the set of agents, A . Hence, $v(S)$ denotes the value (which can be thought of as the level of profit or happiness) derived by the auctioneer if she selects the set S of agents as the auction winners. It is also assumed that when a set S of agents is selected as the winners, these agents are committed to executing the tasks corresponding to their offer (this could be enforced by some penalty mechanism and it is beyond the scope of our work).

To summarize, the auctioneer collects the offers of the agents, and then needs to select a set $S \subseteq A$, as the winning agents, along with a payment to each winner. For this deliverable, we are interested only in the non-strategic version of these problems, where the agents are honest about their cost c_i , and therefore, we only care about the question of determining the allocation of the tasks, i.e., the set of winning agents. Enforcing the agents to be honest about their costs is a topic that has been addressed in WP5 and will therefore not be considered here.

Given the progress that has already been made in the last years for the model that was originally defined by Singer in [3], we define here a somewhat complementary direction to the previous subsection, in which there is no hard budget constraint for the auctioneer (as opposed to [3]). This is very common in several applications where the auctioneer may not have a priori a hard limit on his expenditure. Instead, it is more appropriate to focus on other objectives, which incorporate either the true costs of the agents or the sum of payments by the auctioneer into the objective function. This naturally leads to concentrate on the maximization of the produced utilitarian social welfare, which equals the auctioneer's value minus the total costs of the agents for carrying out the tasks.

To describe more formally the model, we want to optimize the *social welfare* achieved by S , denoted by $SW(S)$, which is the total value derived minus the total cost incurred, expressed as follows:

$$SW(S) = v(S) - \sum_{i \in S} c_i$$

2.2 Cost minimization in spatial crowdsourcing models

The last problem is motivated by spatial models in reverse auctions. There are three main differences compared to the previous two subsections. The first one is that we assume that the tasks lie in some geometric space, i.e., they may correspond to transportation tasks within a Euclidean space. The simplest example is to suppose that the tasks are ordered on a line. The second difference is that we now want to have all the tasks completed. This is unlike Sections ?? and 2.1, where not all tasks are necessarily assigned, either due to the budget restrictions or due to the maximization of social welfare. The third difference is that each task has a demand, on the quality of its completion, and this implies that it may

have to be assigned to multiple agents, to achieve its demand constraint. In applications where it is important to complete all the tasks listed by the auctioneer, it is more natural to focus more on producing a feasible solution of minimum cost. This gives rise to a cost minimization problem that we describe below.

Cost Minimization Interval Cover (CMIC): Consider a set of tasks, say $\{1, \dots, m\}$, that are ordered on a line, and a set of available bidders $A = \{1, \dots, n\}$. An instance of CMIC is determined by a tuple $(\mathbf{b}, \mathbf{q}, \mathbf{d})$, where:

- The vector $\mathbf{b} = (b_1, \dots, b_n)$ is the bidding profile. For each bidder $j \in A$, $b_j = (c_j, [s_j, f_j])$, where $[s_j, f_j] = I_j$ is the interval of contiguous tasks $\{s_j, s_j+1, \dots, f_j\} \subseteq [m]$, that j is able to contribute to, and $c_j \in \mathbb{R}_{\geq 0}$ is the cost incurred to her, if she is selected to contribute. We often denote $\mathbf{b}$ as $(\mathbf{c}, \mathbf{I})$ where $\mathbf{c}$ is the cost vector and $\mathbf{I}$ is the vector of the intervals.
- The vector $\mathbf{q} = (q_1, \dots, q_n)$ is the contribution vector, so that $q_j \in \mathbb{R}_{>0}$ denotes the contribution that bidder j can make to the tasks that belong to $[s_j, f_j]$.
- The vector $\mathbf{d} = (d_1, \dots, d_m)$ specifies the demand $d_j \in \mathbb{R}_{>0}$ of a task j .

The goal is to select a set of bidders $S \subseteq A$ of minimum cost, that satisfies

$$\sum_{i \in \mathcal{N}_j(\mathbf{I}) \cap S} q_i \geq d_j, \quad \forall j \in [m], \quad (1)$$

where for $j \in [m]$, $\mathcal{N}_j(\mathbf{I}) := \{i \in A \mid j \in [s_i, f_i]\}$, i.e., it is the set of bidders who can contribute to task j .

3 Data derivation and acquisition for social welfare maximization

In the following two subsections, we provide a detailed account of the attributes of the datasets, encompassing both synthetic and real data, employed in the experimental assessment of algorithms addressing the optimization problem outlined in Section 2.1.

3.1 Synthetic Data

We will start with the procedure we followed for creating the synthetic datasets. We need to specify for each worker a set of tasks that she is capable to complete. Each task is associated with a valuation from the auctioneer, while each worker incurs a specific cost for performing a subset of tasks. Unique identifiers, such as user ID and skill ID, are assigned to workers and tasks, respectively. Figures 1 and 2 depicts a random dataset instance. That is all the information we need as to which type of data will populate the data set. As we see in the figures, for example, the worker w_{00} needs around 7.018 units to perform tasks t_{00}, t_{06}, t_{04} , and those tasks provide around 18.352, 13.802, 1.969 units respectively¹. The uniqueness of user_id and skill_id ensures distinct identification for users and skills.

Our goal is to set the parameters of the problem in such a way, so that the generated instances are nontrivial. Thus, the function for the social welfare should take both positive and negative values, and also the optimal solution should be neither the empty set $\emptyset$,

¹We assume that these units represent the same value for the cost and the valuations

TASKS DATAFRAME		
skill	skill_id	valuation
t00	0	18.3516558
t01	1	4.06465125
t02	2	16.76146
t03	3	15.1226017
t04	4	13.8019175
t05	5	11.4089937
t06	6	1.96898208
t07	7	16.7048312
t08	8	9.8782273
t09	9	4.79137384

Figure 1: This is a workers sample data frame that we have created

WORKERS DATAFRAME			
cost	skills	user_id	user_name
7.01802501	['t00', 't06', 't04']	0	w00
10.7247216	['t02', 't04', 't07']	1	w01
5.26466198	['t01', 't07']	2	w02
5.225991	['t00', 't07']	3	w03
8.71518532	['t02', 't07', 't05']	4	w04
15.7890548	['t01', 't05']	5	w05
11.7903266	['t00', 't05']	6	w06
12.7648942	['t02', 't05', 't03']	7	w07
27.2393345	['t01', 't03', 't08']	8	w08
29.7004379	['t00', 't08', 't09']	9	w09

Figure 2: This is a workers sample data frame.

nor the whole set A , of the n agents. Therefore we created the instances for the synthetic data by tweaking the following parameters:

- The number of tasks m and the number of workers n .
- The number of tasks per worker.
- The way the tasks are distributed to the workers.
- The values of the valuations per task.
- The values of the costs per worker.

1. In Table 1, we present the chosen combinations for the number of tasks and workers, leading to the creation of 10 datasets. Each dataset encompasses 30 instances generated through three distinct methods (10 instances for each method), which will be described next.

	1	2	3	4	5	6	7	8	9	10
m	10	5	10	20	10	50	100	25	50	100
n	5	10	10	10	20	20	20	50	50	50

Table 1: Datasets for various n and m .

2. For the number of tasks per worker we have used 3 different methods based on 2 distributions. We denote the set of tasks to be completed as $\mathbb{T}$ and the set of tasks an agent i offers as T_i :

- (a) $|T_i| \sim \mathcal{U}(1, \frac{m}{3})$, $\forall i \in \{1, \dots, n\}$ and $m = |\mathbb{T}|$, where $\mathcal{U}$ is the uniform distribution.
- (b) $|T_i| \sim \mathcal{DN}(1, \frac{m}{3})$, $\forall i \in \{1, \dots, n\}$ where $m = |\mathbb{T}|$ and $\mathcal{DN}$ denotes a discrete analogous of the Normal distribution that we have created.
- (c) $|T_i| \sim \mathcal{DN}(1, \frac{m}{3})$ for 95% of the workers and $|T_i| \sim \mathcal{DN}(1, \frac{m}{2})$ for the rest 5% of workers, $\forall i \in \{1, \dots, n\}$ and $m = |\mathbb{T}|$.

The Discrete Normal distribution is generated through the following method:

- (a) The initial interval, adjusted as needed, is partitioned into evenly spaced, distinct sub-intervals, excluding the borders.
- (b) Each sub-interval is constructed in such a way, that only one integer is encompassed within it.
- (c) The probability assigned to the integer contained in the sub-interval corresponds to the probability of that sub-interval, given that the initial interval was derived from the $[-3\sigma, 3\sigma]$ range of a Normal distribution $\mathcal{N}(\mu, \sigma^2)$, where μ is the center of the initial interval.
- (d) The assigned probabilities are normalized to ensure they collectively sum to 1.

This method enables the creation of a discrete analogue of a Normal Distribution.² To facilitate a deeper comprehension of the process, we present an illustrative example.

Example: Given the interval $[1, 10]$.

- (a) The interval is adjusted to $[0, 11]$ and it is divided into 10 sub-intervals as follows: $[0.5, 1.5]$, $[1.5, 2.5]$, $[2.5, 3.5]$, $[3.5, 4.5]$, $[4.5, 5.5]$, $[5.5, 6.5]$, $[6.5, 7.5]$, $[7.5, 8.5]$, $[8.5, 9.5]$, $[9.5, 10.5]$. Notably, sub-intervals are chosen to maintain balance and avoid covering the entire interval.
- (b) The integers 1 through 10 correspond to the respective sub-intervals.
- (c) The initial interval is set as the $[-3\sigma, 3\sigma]$ range of a Normal distribution with μ as the center of the interval. To achieve this, μ is set to 5.5, and both halves are spanned by 3σ . Thus, $(11 - 5.5) = 3\sigma$ yields $\sigma = 1.834$. The resulting normal distribution is $\mathcal{N}(5.5, 1.834^2)$. Probabilities for the sub-intervals are calculated using the cumulative distribution function (CDF), denoted as $p_1, \dots, p_{10}$. The probability assigned to each integer within the initial interval is determined by the formula:

$$pr_i = \frac{p_i}{\sum_{i=1}^{10} p_i}, \quad \forall i \in \{1, \dots, 10\}$$

3. The tasks are categorized into popular and non-popular tasks. Popular tasks constitute one-fourth of the total tasks, while the remaining tasks are considered non-popular. Each worker is assigned 10–25% of their available tasks, drawn from popular and non-popular tasks. Selection is accomplished using a "sliding window with

²This idea was inspired by an answer on <https://stackoverflow.com/> for creating a discrete normal distribution in python.

no cover” for popular tasks and a ”sliding window with cover” (2-3 tasks overlap) for non-popular tasks. For instance, if worker w_1 has 2 tasks from a set of 4 popular tasks (t_1, t_2, t_3, t_4) , she will be assigned t_1, t_2 . The same logic applies to non-popular tasks with overlapping enforced by sliding the window two to three spots to the left each time.

4. Task valuations are chosen uniformly from two super-sets of datasets. One super-set has a low constant upper bound of 20 for the uniform distribution, while the other has an upper bound of m . Valuations for the first super-set are drawn from $\mathcal{U}(1, 20)$, and for the second super-set, they are drawn from $\mathcal{U}(1, m)$. The second super-set accounts for substantial costs to prevent a highly negative objective function.
5. The costs for each instance are determined by the following method:
 - (a) Split workers randomly into sets S_1 , containing 50% of the workers, S_2 containing 35%, and S_3 containing the rest 15%. Notice $S_i \subseteq \mathbb{W}$ and they are also distinct sets $\forall i = 1, 2, 3$.
 - (b) For S_1 , the costs are created uniformly using $\mathcal{U}(|T_i|, 4 \cdot |T_i|)$
 - (c) For S_2 , the costs are created uniformly using $\mathcal{U}(4 \cdot |T_i|, 8 \cdot |T_i|)$
 - (d) For S_3 , the costs are created uniformly using $\mathcal{U}(8 \cdot |T_i|, 12 \cdot |T_i|)$, where $|T_i|$ is the number of task for the i -th worker.

To summarize, two super-sets, each comprising of ten datasets, are generated, with each dataset containing thirty instances of synthetic data. Various methods for creating task sets for each worker are employed to introduce overlaps and enhance the results. The subsequent subsection describes the implementation of real datasets.

3.2 Datasets Based on Real Data

The real data utilized in our study originated from the platforms Guru and Freelancer, employing preprocessed data from [2]. The only information that was not available was the valuations of the auctioneer for each task. Hence, we decided on a methodology to assign valuations to tasks, a process examined in this subsection. The datasets were generated from each super-set (Guru and Freelancer), with the combinations of n and m values chosen for each dataset in a similar way, as presented in Table 1. The selection of these values was informed by the size of each super-set.

The following steps are describing the method for creating the datasets and their instances for each group of real data, guru and freelancer.

1. **Super-set Construction:** The super-set comprises 10 datasets, each containing 30 instances with n tasks and m workers. The combination of n and m varies for each dataset.
2. **Categorization Based on Popularity:**
 - Top 20% of tasks, frequently assigned to workers, are designated as popular tasks.
 - The next 60% are classified as common tasks.
 - The remaining 20% are categorized as rare tasks.

3. **Valuation Assignment:**

- A proper valuation for each task is determined using the Discrete Normal Distribution, as detailed in Section 3.1.
- The upper bound of the distribution is set as double the average valuation of the auctioneer, a value unknown due to missing task valuations.
- A lower bound is defined based on the assumption that the average valuation multiplied by the average number of assigned tasks to each worker should exceed the average cost of each worker.
- The upper bound is established as the average cost of each worker divided by the average number of assigned tasks to each worker.

4. Dataset Generation:

- For each dataset, m random tasks are selected from the super-set, with a ratio of 20% – 60% – 20% for rare, common, and popular tasks.
- We include n random workers in the set, chosen logically to ensure that each worker’s cost does not surpass the sum of valuations for her provided tasks.

In summary, this process yields 10 datasets within each super-set, each comprising 30 instances of synthetic data (based on real data). The methodology encompasses categorizing tasks by popularity, assigning valuations using the Discrete Normal Distribution, and generating datasets with logical worker-task assignments based on cost and valuation considerations.

4 Datasets for spatial crowdsourcing

We describe now the process we used for building datasets for spatial crowdsourcing auctions. Due to the general lack of available data from actual applications, we only used synthetic data for our analysis. We outline first the precise data requirements. As elucidated in Subsection 2.2, the dataset necessitates a collection of tasks, each with specified demand requirements. For every worker, a declared cost, a contribution parameter, and an interval of consecutive tasks she can undertake are essential details. This encapsulates all the necessary information to populate the dataset. To generate instances for the synthetic data, we selected a range of values for the following parameters:

- The number of tasks m .
- The number of workers n .
- The interval I_j for each worker $j \in A$.
- The demand d_i of each task $i \in [m]$.
- The contribution q_j of each worker $j \in A$.
- The cost c_j of each worker $j \in A$.

For reference, consider an instance of CMIC (sourced from [1]) presented in Example 1, which also aids in providing a clearer understanding of both the input and the objective of the problem under study.

Example 1. Suppose we have a set of $m = 5$ cities located geographically in a consecutive order, with a company having a store in each city. The goal for the company is to meet a demand constraint, i.e., a minimum amount of raw materials each store needs to operate. In order to do that, the company hires other firms, or single individuals, to make deliveries, via a reverse auction. Each of the $n = 5$ participating bidders can only cover

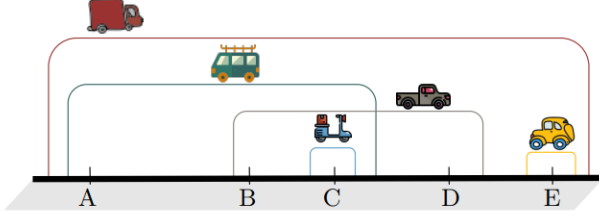


Figure 3: Illustration of the tasks that workers from Example 1 can perform.

vehicle	contribution cost	
truck	8	10
van	6	5
pickup	5	4
motorbike	1	2
car	3	4

Figure 4: Workers' parameters from Example 1

a specific interval of contiguous cities, at a certain cost, with a designated contribution, i.e, the amount of goods it can deliver to each city, assumed to be the same for each city within its declared interval. Our problem comes down to selecting a set of winning bidders at a minimum cost who can jointly cover the demand constraints of each city. For this example, let's assume that the demand of cities A to E are given by the vector (6, 2, 9, 1, 3); the interval that each worker can cover is visible in Figure 3; the contribution per task, as well as the cost of the workers, are given in Figure 4. An obvious feasible solution would be to hire all the workers. However this is not the optimal assignment as it is not minimal. Selecting the workers with the van, the pickup and the car also forms a feasible solution since the demands of all cities are indeed satisfied. Notice that the optimal solution is to hire the workers with the truck and the motorbike at a total cost of 12.

As per the production of specific values for the aforementioned parameters, we use randomized procedures outlined below. A primary consideration is to avoid generating data that lacks any feasible solution. Ideally, instances with substantial disparities between the optimal solution's cost and the costs of the two approximation algorithms are sought to validate worst-case approximation ratios. To make the instances appear natural, we both included workers the costs of which are determined based on their total contribution as well as oblivious workers: outliers with markedly higher or lower costs compared to their contribution. We provide below more details on the parameters' specification.

Tasks: The number of tasks we chose can be seen in the table below. In total, we have generated 8 datasets, each corresponding to a different choice of m . For each dataset, 40 distinct instances of CMIC have been created, varying according to the remaining input parameters.

dataset	1	2	3	4	5	6	7	8
m	10	20	35	50	65	75	85	100

Workers: For each instance, we generated $3m$ *small* workers and w *large* workers, where w is selected uniformly at random from the set $\{0, \dots, \frac{3m}{20}\}$. This classification of workers is based on the number of tasks each worker can undertake, as elaborated below.

Tasks per Worker: To determine the tasks that each of the workers can perform we use the following method:

- For each small worker, we randomly select a task from the available tasks in the instance. Then, this worker is assigned to perform the chosen task, and

in addition to that, she covers an additional number of x tasks to the left of the picked task and y tasks to the right, where x and y are two values chosen at random between $\{0, 1, 2\}$, if $m = 10$, and between $\{0, 1, 2, 3\}$, otherwise. If adjusting x or y goes beyond the task set limits, we modify the selected values to ensure the intervals remain within the bounds.

- For large workers, we also randomly select a task from the available set for them to cover, but now, each worker has a fixed number of tasks they can perform around the chosen task. We aim for a symmetrical distribution of coverage, once again adjusting if the interval extends beyond the specified number of tasks. For each value of m , we use the sizes of intervals presented below:

m	10	20	35	50	65	75	85	100
size of interval	5	8	10	12	14	16	18	20

Once intervals of tasks are assigned to workers, it's possible that certain tasks cannot be fully covered, even if we would select all the workers that are able to perform them. To prevent such trivial instances, we check for any remaining uncovered tasks. If found, we create a new (small) worker for that task, extending coverage to a few additional tasks around it. This process mirrors the one discussed earlier, but instead of choosing a random task, we specifically select an uncovered one. We repeat this until no tasks are left uncovered.

Tasks' Demands: For every task $i \in [m]$, a random value is chosen from the set $\{5, \dots, 20\}$ and is assigned to d_i .

Workers' Contribution Parameters: Let λ_j represent the number of workers capable of contributing to task j . We initiate the process by selecting the first task and assigning each eligible worker a random contribution between 1 and $4d_1/\lambda_1$. After this, we check if the total contribution from these workers is sufficient to cover the task's demand. If yes, we proceed to the next task; if not, we -iteratively and in the same way- assign new random contributions to the workers capable of executing that task, until the demand is met. Before moving to subsequent tasks, we should note that workers able to contribute to it may already have assigned contributions from previous ones. Let Q_j denote the total contribution of the workers able to perform task j who have already received an assignment, considering their capacity for tasks preceding j . Additionally, let λ'_j represent the number of workers who can contribute to task j and are yet to receive a contribution. Define $Q'_j = 2d_j - Q_j$, indicating the remaining total contribution to be assigned to task j . The rationale behind this choice is that, towards achieving feasibility, we aim to specify Q'_j so as to have an instance in which the workers capable of performing task j collectively cover at least double the task's demand. So, for the λ'_j workers, we allocate a contribution randomly chosen between 1 and $\lceil \frac{2Q'_j}{\lambda'_j} \rceil$. Finally, we check if all demands can be covered after all contributions are assigned. If not, we repeat the process, incrementing the upper bound of each randomized selection by 1 each time the solution is not feasible. This includes the assignment of contributions with respect to the first task, as scenarios may arise where some consecutive tasks have exactly the same contributors.

Workers' Costs: To assign costs to the workers, we categorize them into two groups: rational and oblivious. Rational workers make up 80% of the total workforce, with

the remaining workers classified as oblivious. Rational workers receive a random cost between $\left\lfloor \frac{q_i \cdot |(f_i - s_i + 1)|}{3} \right\rfloor$ and $q_i \cdot |(f_i - s_i + 1)|$ while oblivious workers receive a random cost between 1 and $q_i \cdot |(f_i - s_i + 1)|$, where $|(f_i - s_i + 1)|$ represents the number of tasks that worker i can perform and f_i, s_i can be inferred from the discussion on the tasks' assignment.

Overall, we have employed a method that ensures a substantial degree of variation in our datasets, and therefore we believe it produces realistic instances and avoids trivialities.

5 Implementation aspects

5.1 Social welfare maximization

5.1.1 Data Storage and Organization

In this subsection, we will briefly outline the choices made in implementing the experiments for the problem of social welfare maximization. To maintain the need for computational power to a certain level, the largest instance we tested was for $n = 100$ and $m = 400$. Moreover, we opted for the utilization of CSV files for data storage instead of employing a database. Various file parsers and exporters were developed, enabling the immediate creation, reading, and writing of worker and task data frames for each experiment instance. A global configuration file, `config.json`, consolidates all file paths and project structure details, serving as a reference for both parsers and exporters. Throughout the data creation and experiment execution phases, a suite of diverse loggers was employed for debugging and archival purposes. The results are preserved in three distinct formats:

- The dataframes and all experiment parameters are dumped into CSV files (`.csv`) for each instance.
- Crucial result parameters are stored in an Excel file (`.xlsx`) for subsequent analysis in the following chapter.
- The dataframes and experiment parameters are stored as pickled files (`.pickle`), facilitating immediate revisitation without the need for parsing or exporting/converting.

Conclusively, the results for each dataset and instance are stored in designated folders for each algorithm, such as `"DistortedGreedy/Dataset_1/Instance_1"`. Both instances for the two super-sets are housed in the same location. The execution of the experiments for each dataset is straightforward: all algorithms are executed, and the results are diligently stored in the aforementioned files.

5.1.2 Optimization

In this section, we delve into the implementation details of the solvers employed for obtaining the optimal solution for each created instance, so that we can have a benchmark to compare against, for evaluating the performance of the implemented algorithms. This is done by solving an integer program, that we refer to as the Winner Determination Integer Program (WD-IP). At the same time, we also show how we solved a relaxation of this, referred to as the Winner Determination Linear Program (WD-LP), which is utilized in one of the algorithms we implemented (the LP-Based algorithm as reported in Deliverable D.6.2).

1. **Winner Determination Integer Program (WD-IP)** We can define SWM as the following optimization problem.

$$\underset{\mathbf{x}, \mathbf{y}}{\text{maximize}} \quad \sum_{t_i \in \mathbb{T}} v_i x_i - \sum_{w_j \in \mathbb{W}} c_j y_j \quad (2)$$

$$\text{subject to} \quad x_i \leq \sum_{j: w_j \in S} y_j, \quad \forall t_i \in \mathbb{T} \quad (3)$$

$$x_i, y_j \in \{0, 1\}, \quad \forall i: t_i \in \mathbb{T}, j: w_j \in \mathbb{W}, \quad (4)$$

2. **Winner Determination Linear Program (WD-LP)** In this version of the problem we relax the constraints of the problem and allow the binary values to range between 0 and 1.

$$\underset{\mathbf{x}, \mathbf{y}}{\text{maximize}} \quad \sum_{t_i \in \mathbb{T}} v_i x_i - \sum_{w_j \in \mathbb{W}} c_j y_j \quad (5)$$

$$\text{subject to} \quad x_i \leq \sum_{j: w_j \in S} y_j, \quad \forall t_i \in \mathbb{T} \quad (6)$$

$$0 \leq x_i, y_j \leq 1, \quad \forall i: t_i \in \mathbb{T}, j: w_j \in \mathbb{W}, \quad (7)$$

For these implementations, we utilized two Python libraries: cvxpy for WD-IP and SciPY for WD-LP. To enhance efficiency and compatibility with optimization libraries, we transformed the problem by representing the objective function differently.

We start the transformation by using a different representation for the objective function. The expression (5) can be written as follows:

$$\sum_{t_i \in \mathbb{T}} v_i x_i - \sum_{w_j \in \mathbb{W}} c_j y_j = \mathbf{c}^T \mathbf{x}$$

where $\mathbf{c} = \begin{pmatrix} v \\ -c \end{pmatrix}$, $v \in \mathbb{R}^{+m \times 1}$, $c \in \mathbb{R}^{+n \times 1}$, $\mathbf{c} \in \mathbb{R}^{+(m+n) \times 1}$, $\mathbf{x} \in \mathbb{R}^{+(m+n) \times 1}$.

The first m rows of the $\mathbf{c}$ vector represents the valuations for each task, and the rest n rows represents the cost of the workers. Respectively, the first m rows of $\mathbf{x}$ are the x_i dummy variables of the original problem and the last n rows are the y_j dummy variables of the original problem.

For the first inequality constraint (6) we will use the following equivalent matrix inequality:

$$A\mathbf{x} \leq \mathbf{0} \quad A \in \mathbb{R}^{m \times m+n}, \quad \mathbf{x} \in \mathbb{R}^{m+n \times 1}$$

$$A = \begin{pmatrix} 1 & 0 & \cdots & 0 & -y_{1,1} & -y_{1,2} & \cdots & -y_{1,n} \\ 0 & 1 & \cdots & 0 & -y_{2,1} & -y_{2,2} & \cdots & -y_{2,n} \\ 0 & 0 & \cdots & 0 & -y_{3,1} & -y_{3,2} & \cdots & -y_{3,n} \\ \vdots & \cdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -y_{m,1} & -y_{m,2} & \cdots & -y_{m,n} \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n+m} \end{pmatrix}$$

If the j -th worker can perform the i -th task then $y_{i,j} = 1$ else it is equal to 0. As an example of 3 tasks and 4 workers

$$A_{ex} = \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 & -1 & 0 \end{pmatrix}$$

represents the matrix where the first worker can perform the 1st and 3rd task, the second can perform the 2nd task, the third worker can perform all three tasks and the 4th worker can perform no tasks.

Finally for the last inequality constraint (7) we will use the following constraint: $0 \leq x_i \leq 1$ for $x_i \in \mathbf{x}$. So the problem WD-LP takes the final form:

$$\underset{\mathbf{x}}{\text{maximize}} \quad \mathbf{c}^T \mathbf{x} \quad (8)$$

$$\text{subject to} \quad A\mathbf{x} \leq \mathbf{0} \quad (9)$$

$$0 \leq x_i \leq 1, \text{ for } x_i \in \mathbf{x} \quad (10)$$

The WD-IP problem is exactly the same with the only difference being the second inequality constraint, that is changed to $x_i \in \{0, 1\}$. So the WD-IP is defined as follows:

$$\underset{\mathbf{x}}{\text{maximize}} \quad \mathbf{c}^T \mathbf{x} \quad (11)$$

$$\text{subject to} \quad A\mathbf{x} \leq \mathbf{0} \quad (12)$$

$$x_i \in \{0, 1\}, \text{ for } x_i \in \mathbf{x} \quad (13)$$

This comprehensive analysis covers the processes and mechanisms employed for conducting experiments, encompassing both the theoretical formulation of the optimization problems and their practical implementation.

5.2 Spatial Crowdsourcing

In this subsection, we briefly outline the choices made for the data generation process. We conducted experiments using synthetic datasets that we created specifically to mimic real-world data. All the experiments were performed on a Windows 10 machine and each of them run for a relatively short period of time. Throughout the experiments, the primary programming language that was used was Python, and many standard libraries of it were utilized. The upper bounds on the number of workers, n , was set as discussed in Section 4, primarily for implementation purposes, as higher values of n were observed to considerably slow down our experiments. We also employed the specified upper bounds on the number of tasks, since instances with a significantly higher number of tasks appeared to be rather unrealistic.

5.2.1 Data Storage

As in Section 5.1, to keep things simple, we decided to use CSV files over a database for storing both the data and the results. For the data, we kept essential parameters such as the bidding profile, contribution, and the demand vector. In terms of results, we kept key parameters, including the id of the dataset used in each case, the number of workers, the number of tasks, the maximum number of workers that can perform each task as well as the task that achieves the maximum such value, and the solutions generated by the tested algorithms along with their costs. To execute the data generator, we employed a parametric design approach, simply providing the IDE with the necessary parameters each time. For the multiple dataset requirements, we had to alter the seed of the randomized library each time we wanted to generate multiple datasets. Finally, we keep a file containing the results of all algorithms, as well as separate files including the results for instances with the same number of tasks, such as `Results > results_for_10_tasks`.

5.2.2 Optimization

Now we delve into the implementation details of the solvers employed for obtaining the optimal solutions for CMIC in each created instance. This is needed so that we can compare the performance of our approximation algorithms against the optimal solution, which is the ideal benchmark. The procedure we used is based on a formulation of the studied problem as an Integer Linear Program (ILP) and helps us determine the optimal solution of a given instance towards comparing it with the solutions obtained by the studied algorithms. According to Subsection 2.2, the formulation is as follows:

$$\begin{aligned} & \text{minimize } \sum_{i \in A} c_i x_i \\ & \text{subject to } \sum_{i \in \mathcal{N}_j(\mathbf{I})} q_i x_i \geq d_j, \quad \forall j \in [m], \\ & \quad x_i \in \{0, 1\}, \quad \forall i \in [n], \end{aligned}$$

where for $j \in [m]$, $\mathcal{N}_j(\mathbf{I}) := \{i \in A \mid j \in [s_i, f_i]\}$, i.e., it is the set of bidders who can contribute to task j .

In the experiments, to find the optimal solution of an instance using the ILP formulation presented above, we utilized the PuLP library, which is an LP modeler for Python that can efficiently solve LPs. Our approach takes an instance of CMIC as input and from that it creates a binary decision variable for each worker, defines the objective function and sets up constraints based on the demand and contribution values, as suggested by the constraint of the program. Then, it solves the ILP optimally by determining a feasible set of workers that minimizes the total cost.

6 Conclusions and Future Plans

We have outlined our methodology for creating synthetic data sets for the problems under consideration. We also exploited the availability of real world data in the context of the social welfare maximization problem. The next step in project COMRADE is to provide an experimental evaluation of the relevant algorithms that we have implemented for the problems described in Section 2. The implemented algorithms and the results of the experimental evaluation will be reported as part of Deliverable D.6.2.

References

- [1] Evangelos Markakis, Georgios Papasotiropoulos, and Artem Tsikiris. On improved interval cover mechanisms for crowdsourcing markets. In *International Symposium on Algorithmic Game Theory*, pages 94–112. Springer, 2022.
- [2] Sofia Maria Nikolakaki, Alina Ene, and Evimaria Terzi. An efficient framework for balancing submodularity and cost. 2021.
- [3] Yaron Singer. Budget feasible mechanisms. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*,, pages 765–774, 2010.