Algorithmic Game Theory

231847 Paola Guarasci, 231847 Francesca Murano

Department of Mathematics and Computer Science,

University of Calabria

University Project

Research Objective

Define a system that can choose many users to take a tour. The number of users and the maximum number of kilometers that can be traveled is fixed. Each user has a budget that they cannot exceed. The costs of the tour are divided into proportional costs related to the distance of the tour and fixed costs.

Outline

1. Bayesian Game Settings

2. Quasilinear mechanism

2.1. Direct quasilinear mechanism

3. Vickrey-Clarke-Groves Mechanisms

4. Problem Resolution

4.1. Definition of variables

4.2. Overview resolution

4.3. Example Application

4.4. Code implementation

1 Bayesian Game Settings

A Bayesian game is a type of game in which players have **incomplete information** about the other players' characteristics or strategies. In this game, each player has a prior belief about the distribution of the other players' types.
The main challenge in a Bayesian game is to find a strategy that maximizes a player's expected **utility** given their prior beliefs about the other players. They are called Bayesian because of the probabilistic analysis inherent in the game.
A Bayesian game is a tuple (N,O,Θ,p,u), where:

- N = {1,...,n} is a finite set of players;
- O = a set of outcomes;
- $\Theta = \theta_1 \times \cdots \times \theta_n$, with θ_i the set of possible types of player i;
- p : $\Theta \rightarrow [0,1]$ is the probability distribution over Θ;
- u = $(u_1, ..., u_n)$ is a profile of utility functions u_i : $A \times \Theta \rightarrow R$.

2 Quasilinear Mechanism

A mechanism in the quasilinear setting is a triple (A, χ ,$\mathcal{P}$), where:

- $A = A_1 \times \cdots \times A_n$, where A_i is the set of actions available to agent $i \in N$;
- x : $A \rightarrow \Pi(X)$ maps each action profile to a distribution over choices;
- $\mathcal{P}$: $A \rightarrow R^n$ maps each action profile to a payment for each agent.

The use of a quasilinear mechanism came about because its strengths:

- the mechanism can choose to charge or reward agents with an arbitrary monetary amount, so our users can choose their own budget;
- an agent's degree of preference for selecting **any choice is independent** of the amount. So our users will choose the destination of the trip just for pleasure without thinking about the amount needed. This feature is really important as it leads users to focus only on their own interests without seeking strategies based on others' choices. They have no need to lie and always proclaim the truth.

2.1 Direct quasilinear mechanism

A direct quasilinear mechanism is a pair (χ,$\wp$). It defines a mechanism in the quasilinear setting, where for each i, $A_i = \Theta_i$. The set of actions available to each player is just the set of possible preferences of the player.

3 Vickrey-Clarke-Groves Mechanisms

For the problem resolution, the choice fell on a mechanism belonging to the **Groves mechanisms**. Groves mechanisms belong to the direct quasilinear mechanisms (χ,$\wp$),for which:

- $\chi(\hat{\mathrm{v}}) = \mathrm{argmax_x} \sum_{\mathrm{i}}^{\mathrm{n}} \mathrm{v_i(x)}$;
- $\wp_{\mathrm{i}}(\hat{\mathrm{v}}) = h_i(\hat{v}_{\text{-i}}) - \sum_{i \neq j}^{n} \hat{v}_j(\chi(\hat{v}))$;

The dominant strategy of the mechanism is **truthtelling**, this mechanism allows us to make efficient choices.

What does it mean efficiency?

A quasilinear mechanism is efficient, if in equilibrium it selects a choice x such that:

$$\forall v \forall x', \quad \sum_{i=0}{}_{i}^{n} v_i(x) \quad \geq \quad \sum_{i=0}{}_{i}^{n} \quad v_i(x')$$

In VCG mechanism, h$_i$ is replace with:

- $h_i(\hat{v}_i) = \sum_{i \neq j}^{n} \hat{v}_j \ (\chi(\hat{v}_{-i}))$

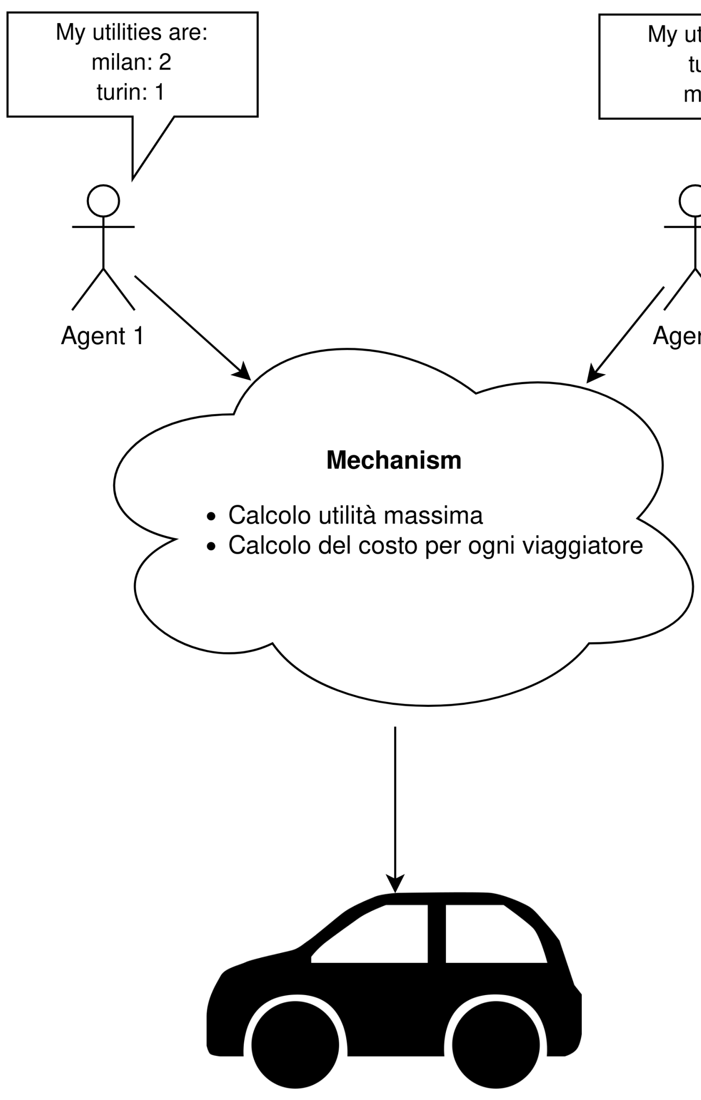
Using this information, it is possible now to establish how the VCG mechanism is structured:

- $\chi(\hat{v}) = \arg\max_x \sum_i^n v_i(x)$;
- $\wp_i(\hat{v}) = \sum_{i \neq j}^{n} \ \hat{v}_j(\chi(\hat{v}_{-i})) - \sum_{i \neq j}^{n} \ \hat{v}_j \ (\chi(\hat{v}))$;

4 Problem Resolution

Goal: Organize a travel composed by k players following some rules:

- each player has a budget;
- each player is an agent represented by an arc;
- each player has fixed and variables costs proportional to travel length;
- each player declares truthfully its interests;
- the bus has a fixed number of places;
- the tour start and finish in the same place;
- the bus has maximum number of km that can be covered;



4.1 Definition of variables

- P :- set of total players;

$$P = \{\text{Francesca, Paola}\}$$

- S :- subset of players joining the travel;

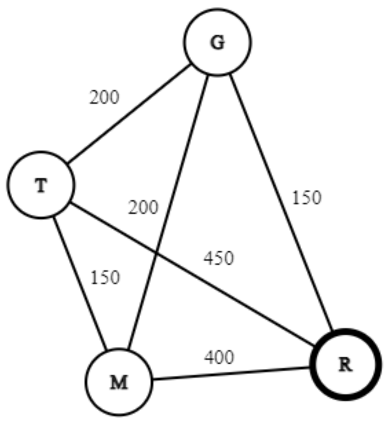
$$S = \{S \subseteq P\text{: } |S| = k\} \text{ , k is the number of players;}$$

- L :- set of locations available;
- L_S = start location;
- T_P :- set of tour's path;
- P_U :- set of players' utilities;
- MAX_L :- limit of kilometers;
- D_L :- set of all distances for each place.
- B_i :- Budget maximum for each player.

4.2 Overview Resolution

1. We generate all possible path.
2. We apply the "traveling salesman problem", we search the shortest path of tour and eliminate all path that exceed the limit (MAX_L).
3. Calculate the sum of utilities for each path.
4. We apply the mechanism VCG to maximize first of all the utilities of tour and than apply costs to all players.
5. We repeat the process if costs exceed the budget.
6. If the costs collected by the mechanism exceed the cost of tour, we divide it among the participants according to the kilometers traveled.

4.3 Example Application



- P = S = {Paola, Francesca};
- k = 2;
- B_{Paola} = 500€;
- $B_{\text{Francesca}}$ = 500€;
- L = {Rome, Genoa, Milan, Turin};
- D_L = { Rome-Turin: 670km, Rome-Milan: 346km, Rome-Genoa: 500km, Milan-Turin: 143km, Milan-Genoa: 142km, Turin-Genoa: 172km };
- DP_L = { Rome-Turin: 450€, Rome-Milan: 400€, Rome-Genoa: 150€, Milan-Turin: 150€, Milan-Genoa: 200€, Turin-Genoa: 200€};
- L_S = Rome;
- P_U = { Paola = [M=0,G=1,T=2], Francesca = [M=0,G=2,T=1] }
- MAX_L = 1600km;

1. Generation of all possible combination without repetition (who started from Rome).

$$T_P = [RGMT, RG, RM, RT, RMG, RGT, RMT];$$

2. Result of Salesman problem:

$$T_P = [RGMT, RG, RM, RT, RGM, RGT, RMT]$$

3. Calculate path's distance and eliminate that do not respect the constraint (we have to multiply the distance by two because we consider round trip).

- **RT** = 670 km * 2 = 1340 km;
- **RM** = 346 km * 2 = 692 km;
- **RG** = 500 km * 2 = 1000 km;
- **RGT** = [(RG) 500 km + (GT) 172 km] * 2 = 672 km * 2 = 1344 km;
- **RGM** = [(RG) 500 km + (GM) 142 km] * 2 = 642 km * 2 = 1284 km;
- **RMT** = [(RG) 500 km + (MT) 143 km] * 2 = 643 km * 2 = 1286 km;
- **RGMT** = [(RG) 500 km + (GM) 142 km + (MT) 143 km] * 2 =
  = 785 km * 2 = 1570 km;

4. Calculate utilities in descending order:

   - **RGT** = 6;
   - **RT** = 3;
   - **RG** = 3;
   - **RGM** = 3;
   - **RMT** = 3;
   - **RM** = 0;

5. We select the first with maximum utilities: **RGT**.

6.

$$\chi(\hat{v}) = \text{arg max}_x \textstyle\sum_i^n v_i(x);$$

   Places considered = **[R G T]**.
   Paola competes for **GT** and Francesca for **RG**.
   Each agent represents the arc of the major preference in the trip.
   G T cities must be crossed, and I am obliged in doing so.
   If an agent (arc) is missed, an alternative route (if possible) is sought that crosses the two cities and minimizes costs [Costs are the weight on the arches].

7.

$$\wp_i(\hat{v}) = \textstyle\sum_{i\neq j}^n \hat{v}_j(\chi(\hat{v}_{-i})) - \sum_{i\neq j}^n \hat{v}_j\ (\chi(\hat{v}));$$

   Brief Formula explication = Cost of the alternative tour without player's participation (and consequently without his arch) - Cost of the tour with player's participation but without your financial contribution.

   - C_P = (RGMT) - (RGT - GT) =
     = 500€ - (350€ - 200€) = 500€ - 150€ = 350€ < B_{Paola}.

- C_F = RMGT - (RGT - RG) =
  600 € - (350€ - 150 €) = 600€ - 200€ = 400€ < $B_{Francesca}$.
- $C_T = C_P + C_F$ = 350€ + 400€ = 750€.

8. The budgets were respected however, there is a surplus that we are going to divide according to the kilometers of the destinations that were competed for.

   - Surplus = C_T 750€ - (RGT) 350€ = 400€;
   - C per km = Surplus / D_{L_i} = 400 € / 672 km (RG + GT) = 0.59€ / km;
   - S_F = C per km * RG km = 0.59€ * 500 km = 296€;
   - S_P = C per km * GT = 0.59€ * 172km = 104€;
   - C_P - Surplus = 350€ - 296€ = 54€;
   - C_F - Surplus = 400€ - 104€ = 296€;

4.4 Code implementation

The implemented code is divided into several stages::

- Calculation of all possible tours with their respective costs.
- Removal of tours that do not respect the imposed kilometer limits.
- Calculation of utilities for each tour.
- Selection of the tour that maximizes the utility.
- Arc-agent assignment.
- Calculation of costs and surplus allocation.

```
calculateAllTourWithCost(["milano", "torino", "genova", "roma"])

[(('torino', 'genova', 'roma'), 1342), (('torino', 'roma'), 1340), ((' 
   milano', 'torino', 'genova', 'roma'), 1330), (('milano', 'torino', '
   roma'), 1159), (('genova', 'roma'), 1000), (('milano', 'genova', '
   roma'), 988), (('milano', 'roma'), 692)]
```

With this function, we obtain all possible tours from a list of locations. We begin by calculating the set of parts of the list passed as an argument, from this set of all possible groupings of locations we then subtract the sets with dimensions less than 2. We then proceed to calculate the kilometer distance for each of the remaining tours.

```
removeTourOverKMLimit(allTour, const.MAXKMFORTOUR)
 [(('milano', 'torino', 'roma'), 1159), (('genova', 'roma'), 1000), (('
   milano', 'genova', 'roma'), 988), (('milano', 'roma'), 692)]
```

This results in a filtered list of tours, i.e. without tours exceeding the maximum number of km imposed as a constraint in the track.

```
calculateUtilityForAllTour(tours, agents)
 [(('milano', 'torino', 'roma'), 4), (('milano', 'genova', 'roma'), 4),
   (('genova', 'roma'), 2), (('milano', 'roma'), 2)]
```

For each tour in the set of tours passed as a parameter, it calculates the total utility by summing up the utility of the agents for each of the cities in the tour under consideration.

```
selectedTour = allTourCappedWithUtility[0]
```

Let's select the tour. We can take the first item in the list because the `calculateUtilityForAllTour` function returns a list sorted by utility, in reverse order. The first item in the list, therefore, maximizes the utility.

```
for agent in agents:
  agent.getCityWithUtility(selectedTour)
```

Here we highlight the cities of interest (with `utility > 0`) for each agent within the selected tour.

```
calcoloIncassoTotale(selectedTour, agents)
```

The total collection calculation is a sum of the individual payments calculated for each agent. Below is the cost calculation function for each individual agent.

```python

def calculateAgentPayment(selectedTour, agent):
    agentInterest = agent.getCityMaxUtility(selectedTour)
    cityAlternative = []

    for city in selectedTour:
        if city not in agentInterest and city != const.STARTPLACE:
            cityAlternative.append(city)

    agentInterest.append(const.STARTPLACE)
    cityAlternative.insert(0, const.STARTPLACE)
    cityAlternative.append(const.STARTPLACE)
    if len(cityAlternative) > 1 and len(agentInterest) > 1:
        _, costoPercorsoAlternativo = cityNetMoney.findAlternative(
    cityAlternative)
        _, costoTourScelto = cityNetMoney.
    findShortestPathBetweenAllSelectedLocations(selectedTour)
        _, costoTrattaInteresse = cityNetMoney.findAlternativeAB(
    selectedTour, agentInterest[0][0], agentInterest[0][1])
        _, distanzaKmTrattaInteresse = cityNetKm.findAlternativeAB(
    selectedTour, agentInterest[0][0], agentInterest[0][1])
        agent.kmTrattaInteresse = distanzaKmTrattaInteresse
        costo = costoPercorsoAlternativo - (costoTourScelto -
    costoTrattaInteresse)
        return (costo, agent.budget < costo)
    return (0, True)
```

`cityNetMoney` and `cityNetKm` are two graphs identical as arcs and as nodes, varying only as weights on the arcs. In the former the weights represent the `costs in euros`, in the latter, they represent the `km`. On both of them the function `findShortestPathBetweenAllSelectedLocations(list_locations)` uses the TPS algorithm to calculate the shortest path between the locations passed as arguments. `findAlternative(lista, a, b)` uses Dijkstra's algorithm for the cost calculation between A and B.

The cost function then calculates:

- `costoPercorsoAlternativo`, cost of the alternative route without the agent (i.e. the route needed to complete the tour but without the agent's preferred city).
- `costoTourScelto`, cost of the chosen tour (without modification)
- `costoTrattaInteresse`, cost of the route of interest of the specific agent and returns in the output the final cost calculated as

- `costoPercorsoAlternativo - (costoTourScelto - costoTrattaInteresse)`.

```
costoRealeTourMoney = getTourCostMoney(selectedTour)
surplus = incassoTotale - costoRealeTourMoney
costoRealeTourKm = getTourCostKm(selectedTour)
costoAlKm = surplus / costoRealeTourKm
```

Here any surplus is calculated, the VGC is not budget-balanced! The mechanism could collect more money than is needed to do the tour, thus in `surplus` precisely. The `surplus` we decided to use as a discount on the tour fee, inversely proportional to the distance traveled by the tour participants to meet the choice of the agent whose payment is being calculated.

```
  for agent in agents:
    print(agent.name)
    calcoloScontoEPagamentoFinale(agent, costoAlKm)
```

Finally, the surplus and final payment calculation function returns the net amount to be paid for each agent.