Project book

Optimization-based Mechanisms for the Course Allocation Problem

Moriya Ester Ohayon, Tamar Bar-Ilan & Ofek Kats

project advisor: Prof. Erel Segal-Halevi

content

Introduction

Course Allocation Problem (CAP)

In the course allocation problem, a university administrator seeks to efficiently and fairly allocate seats (or items) in over-demanded courses among students (or agents) with heterogeneous preferences.

Fairpyx- Python library containing various algorithms for fair allocation, with an emphasis on Course allocation.

Algorithms we added to the Fairpyx Library:

TTC (Top Trading-Cycle): Assigns one course in each round to each student, the winning students are defined based on the students' bid values.

SP (Second Price): In each round distributes one course to each student, with the refund of the bids according to the price of the course.

TTC-O: Assigns one course in each round to each student, the winning students are defined based on the students' bid values. Uses linear programing for optimality.

SP-O: In each round distributes one course to each student, with the refund of the bids according to the price of the course. Uses linear programing for optimality.

OC (Ordinal-then-Cardinal): In the OC algorithm for CAP, we maximize ordinal utility followed by maximizing cardinal utility among rank-maximal solutions, performing this two-part optimization once for the whole market.

Based on "Optimization-based Mechanisms for the Course Allocation Problem", by Hoda Atef Yekta, Robert Day (2020, https://doi.org/10.1287/ijoc.2018.0849)

GitHub Code: https://github.com/Final-Project-fairpyx/fairpyx/

Demo website: https://tome.csariel.xyz/

Project steps:

1. Summary of the Article
2. Created Manual Examples
3. Conducted Code Tests
4. Fully Implemented the Algorithms
5. Compared Performance
6. Enhanced Performance
7. Submitted Pull Request
8. Created Demo Website
9. Project Book

Article Summery

Introduction:

In recent years, many universities have adopted an automated approach for fair course allocation using an algorithm based on a bidding system. The method of converting points into student schedules is well-documented in computer science literature. This article presents five algorithms and compares them based on categories such as efficiency and fairness.

- Efficiency Measures: Efficiency in allocation refers to the number of courses allocated to students. The goal is to ensure that no student's situation can be improved without worsening the situation of another student.
- Fairness Measures: Fairness is assessed by ensuring that there are no significant disparities in utility between students.

The Algorithms:

The TTC (Top Trading Cycle) and SP (Second Price) algorithms operate on an auction-based system. Each student receives a certain number of points to distribute among the courses as they see fit. In each round, the algorithm conducts a kind of "auction," where the student who allocates the most points to a course wins it. Each of these methods also has an optimization process within the rounds that focuses more on fairness.

The OC algorithm, on the other hand, does not operate in rounds but optimizes across the entire market. It performs well in terms of both efficiency and fairness.

In the following sections, we will discuss the simulation results comparing these five algorithms with two common methods (Draft, BPM) using the measures of efficiency and fairness.

BPM (Bidding Point Mechanism):

In this algorithm, each student allocates points to each course, with more points given to the courses they desire more, and fewer points to those they desire less. The algorithm then creates a sorted list based on the points submitted by the students, where the student who allocated the most points to a particular course is placed at the top of the list. The algorithm proceeds through the list, allocating the course to the student who offered the most points, provided there is still space available in the course. As we will see in the following graphs, this algorithm performs poorly compared to the others across most metrics.

Draft:

This algorithm operates in a round-based manner, where each student distributes their points among the courses they wish to take. The algorithm generates a random list of students and assigns their desired course based on the order in the list. In each even round, the order of students is reversed, making this mechanism fairer than BPM. However, it still lacks equality among students since the mechanism does not differentiate between minor and major preferences.

There are two more algorithms that presented in the article:

Random Serial Dictatorship:

In this algorithm, the list of students is arranged randomly, and each student, in turn, receives the full set of desired courses from the algorithm. This algorithm completely prevents strategic behavior but does not consider efficiency or fairness. Typically, this algorithm serves as a foundation upon which significant improvements in fairness and efficiency are added.

GS (Gale–Shapley):

Each student prepares two lists: one ranking the courses and another assigning points to each course (the two lists do not necessarily have to match). In this algorithm, each student bids on the highest-ranked course (that they have not yet received and that still has available seats) from their ranking list. The algorithm then allocates seats (based on the number available) to the students who submitted the highest bids. If the two lists align with each other, the algorithm will produce the same result as BPM. However, this algorithm encourages strategic behavior, and therefore, it is not utilized in this paper.

Additional Research Streams on the Course Allocation Problem (CAP):

In recent years, two additional research streams have emerged regarding the CAP. The first stream describes the problem as a two-sided issue, where courses also have preferences (e.g., preferring students with the highest grades). The existing solutions for this problem do not address overlapping courses. The second stream views the CAP as a supply and demand problem. This article focuses on the CAP according to the second stream.

Notation:

- $\boldsymbol{C}\{\mathbf{1}, \mathbf{2}, \dots, \boldsymbol{m}\}$: The set of courses (there are m courses). A course offered at different times is considered as different courses.

- **j** : An index ranging from 1 to m, representing the specific course.

- **q** : The number of seats allocated to a course (denoted as q_j to indicate the number of seats in course j).

- $\boldsymbol{I}\{\mathbf{1}, \mathbf{2}, \ldots, \boldsymbol{n}\}$: The set of students assigned to course j.

- **k** : The number of courses each student can enroll in per semester (in this paper, k is constant, but in practice, a solution needs to be found for a variable k).

- $\boldsymbol{U_{ij}}$: The utility that course j provides to student i.

- $\boldsymbol{b_{ij}}$: The bid submitted by student i for course j. The total bids from student i for all courses must not exceed a predetermined amount (usually 1000).

- $\boldsymbol{r_{ij}}$: Automatically organizes the list of courses (according to b) in descending order such that: $\forall j, j', r_{ij} \leq r_{ij'} \Leftrightarrow b_{ij} \leq b_{ij'}$

- $\boldsymbol{S_i}$: The set of courses assigned to student i such that there are no scheduling conflicts, and the set contains at most k courses.

- $\boldsymbol{x_{ij}}$: A binary variable that is 1 if student i receives course j, and 0 otherwise.

- $\boldsymbol{O_{jj'}}$: A binary variable that is 1 if there is a conflict between courses or if it is the same course offered at different times. This variable can be defined as transitive (though this may not always be true, so it's an option, not a necessity).

- **t**: The index of the round (for example, $\boldsymbol{x_{ijt}}$ will be 1 if student i receives course j in round t).

- $\boldsymbol{E_{it}}$: The set of all courses available to student i in round t (available meaning a bid has been placed on them and there are seats available). This set changes in each round accordingly.

- $\boldsymbol{p_{jt}}$: The price of course j in round t.

Utility Metrics:

1. Total Cardinal Utility: The total amount the student has bid for the courses they have received.
2. Total Ordinal Utility: The sum of the positions in the list of the courses the student has received (the least desired course is ranked 1).
3. Total Binary Utility: The number of courses the student has received.

Algorithms We Will Work With:

TTC Algorithm (Top Trading Cycle)

Each student submits bids on all courses within a certain budget (up to 1000 points). Each student "votes" for the course on which they have placed the highest bid (from the set of courses available to them). Each course is allocated to up to q_{it} students who have made the highest bids for that course and voted for it. At the end of the round, the course seats are updated based on the number of courses the student has obtained, and the set of available courses for each student changes as follows: from the set of courses available in round t, the course they voted for in round t is removed (even if they did not receive it). If the student did receive the course, all overlapping courses (according to O_{jj}, are removed, as well as all courses that have reached full capacity. In each round, if there are students who have been rejected and did not receive the course they voted for, the algorithm runs another round only with these students until all have received a course in the current round, or no student has a course left in E_{it} . In the next phase, the algorithm runs a round with all students again.

SP Algorithm (Second Price)

This algorithm is very similar to the TTC algorithm but with a point refund mechanism. In each round, each student votes for the first course on their list of available courses. The algorithm allocates the courses to at most q_{it} students who have made the highest bids. If there is enough space for all, the course is considered "free," and the points they bid on it are transferred to the next course on each student's list. If at least one student does not get a place, the course price is updated to the highest amount bid by the rejected students. The students who did get a place pay the new price, and the remaining points are transferred to the next course on their list. The students who were rejected do not pay for the course at all, and all the points from that round are transferred to the next course on their list. In each round, if there are students who have been rejected and did not receive the course they voted for, the algorithm runs another round only with these students until all have received a course in the current round, or no student has a course left in E_{it}. In the next phase, the algorithm runs a round with all students again.

Comparison of TTC and SP:

Both algorithms are equivalent in terms of the total number of courses allocated and the selection of higher-ranked courses on the list.

- In terms of ordinal fairness, the SP algorithm is preferred.

- In terms of cardinal fairness, the TTC algorithm is preferred.

Here is a detailed explanation of how the optimization is applied:

TTC-O Optimization

The TTC-O algorithm operates similarly to the standard TTC algorithm but includes an additional optimization step during each round by defining objective functions and constraints as follows:

1. Objective Function (Z_{t1})

$$Z_{t1} = \text{Max} \sum_{j,i \in E_{it}} r_{ij}\, x_{ijt},$$

This equation maximizes the sum of the product of the rank r_{ij} and the binary variable x_{ijt}, where x_{ijt} is 1 if student i is allocated course j in round t, and 0 otherwise- maximizes the rank of the given courses.

2. Constraint 1:

$$\sum_{i} x_{ijt} \leq q_{jt} \quad \forall j,$$

This constraint ensures that the number of students assigned to course j in round t does not exceed the number of available seats q_{jt}.

3. Constraint 2:

$$\sum_{j \in E_{it}} x_{ijt} \leq 1 \quad \forall i,$$

This constraint guarantees that each student i is allocated at most one course in round t.

4. Binary Variable:

$$x_{ijt} \in \{0, 1\}, x_{ijt} \in E_{it},$$

This defines x_{ijt} as a binary variable, indicating whether student i is allocated course j in round t (1) or not (0).

5. Objective Function (Z_{t2}):

$$Z_{t2} = \text{Max} \sum_{j,i \in E_{it}} b_{ij}\, x_{ijt},$$

This equation maximizes the sum of the product of the bid b_{ij} and the binary variable x_{ijt} - maximizes the bids of the given courses.

6. Equality Condition:

$$\sum_{j,i\in E_{it}} r_{ij}\, x_{ijt} = Z_{t1}.$$

This condition ensures that the sum of the products of ranks and binary variables equals the maximum value Z_{t1}.

In summary, the TTC-O algorithm adds an optimization step to the standard TTC algorithm by maximizing the utility functions while adhering to specific constraints. This approach helps to achieve a more efficient and fair allocation of courses to students.

SP-O Optimization

In the SP-O (Second Price Optimization) algorithm, optimization is performed on course prices during each round by defining the following objective functions and constraints:

1. Objective Function (W_{t1}):

$$W_{t1} = \text{Min}\left(Z_{t1}D + \sum_{j} q_{jt}\, p_{jt} + \sum_{i} v_{it}\right),$$

This equation minimizes the total cost function, which includes the product of the objective function Z_{t1} and the decision variable D, the sum of the products of available seats q_{jt} and course prices p_{jt}, and the sum of v_{it} terms for each student i.

2. Constraint 1:

$$p_{jt} + v_{it} + r_{ij}\, D \geq b_{ij},$$

This constraint ensures that the sum of the course price p_{jt} , the term v_{it} , and the product of the rank r_{ij} and the decision variable D is at least as much as the bid b_{ij}submitted by student i for course j.

3. Constraint 2:

$$p_{jt}, v_{it} \geq 0, D \text{ is unrestricted},$$

This constraint restricts the course price p_{jt} and the term v_{it} to be non-negative, while the decision variable D is unrestricted in sign.

4. Objective Function (W_{t2}):

$$W_{t2} = \text{Min} \sum_{j} q_{jt}\, p_{jt},$$

This equation minimizes the total sum of the products of available seats q_{jt} and course prices p_{jt} .

5. Equality Condition:

$$Z_{t1} D + \sum_{j} q_{jt}\, p_{jt} + \sum_{i} v_{it} = W_{t1}.$$

This condition ensures that the total value of the cost function equals the minimized value W_{t1} .

In summary, the SP-O algorithm optimizes course prices during each round to achieve an efficient allocation of courses by minimizing the cost function while satisfying the constraints related to bids and ranks. This approach allows for a more fair allocation process compared to the non-optimized version of the SP algorithm.

It is important to note that in order to perform the optimization for the SP algorithm (SP-O), the optimization for the TTC algorithm (TTC-O) must first be completed - Z_{t1} from the computation of TTC-O is needed in order to compute SP-O.

This sequential dependency arises because the SP-O algorithm builds upon the structure and results of the TTC-O optimization. The TTC-O process establishes a foundation by optimizing the allocation based on ranks and bids, which the SP-O algorithm then refines by optimizing the course prices. Consequently, the successful execution of SP-O relies on the prior completion of TTC-O, ensuring that the overall allocation process is both efficient and fair.

OC Algorithm (Ordinal-then-Cardinal Optimization)

The OC (Optimization-Based Comprehensive) algorithm optimizes both TTC and SP algorithms in a comprehensive manner, rather than round by round, by defining the following objective functions and constraints:

1. Constraints:

$$\sum_i x_{ij} \le q_j \qquad \forall j,$$

This ensures that the total number of students allocated to course j does not exceed the available seats q_j .

$$\sum_j x_{ij} \le k \qquad \forall i,$$

This constraint ensures that each student i can enroll in at most k courses.

$$x_{ij} + x_{ij'} \le 1 \quad \forall j, j' \text{ with } O_{jj'} = 1,$$

This constraint ensures that if two courses j and j' overlap (or are the same course offered at different times), a student can be enrolled in at most one of them.

$$x_{ij} \in \{0, 1\} \qquad \forall i, j.$$

This defines x_{ij} as a binary variable, indicating whether student i is enrolled in course j (1) or not (0).

2. Objective Function (Z1):

$$Z_1 = \text{Max} \sum_{i,j} r_{ij}\, x_{ij},$$

This maximizes the sum of the product of the rank r_{ij} and the binary variable x_{ij} , optimizing the allocation based on student preferences.

3. Objective Function (Z2):

$$Z_2 = \text{Max} \sum_{i,j} b_{ij}\, x_{ij},$$

This maximizes the sum of the product of the bid b_{ij} and the binary variable x_{ij} , optimizing the allocation based on the bids submitted by the students.

4. Equality Condition:

$$\sum_{i,j} r_{ij}\, x_{ij} = Z_1.$$

This condition ensures that the sum of the products of ranks and binary variables equals the maximum value Z_1 .

In summary, the OC algorithm optimizes the course allocation process comprehensively by simultaneously considering the rank-based and bid-based objectives, subject to constraints that ensure fairness and feasibility. This approach leads to a globally optimized allocation rather than an iterative, round-based process like TTC and SP.

Simulations and Comparisons:

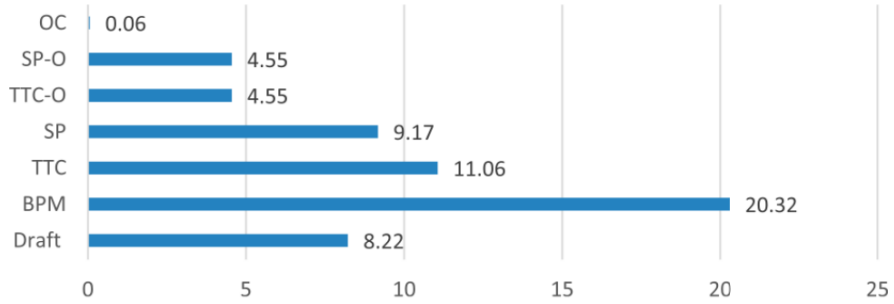
They run the simulation 100 times. Each simulation includes 900 students and 83 available courses. In total, there are 112 course instances (with 19 courses repeating multiple times).

The number of seats in each course for each simulation is drawn based on the following distribution: 10%, 30%, and 60% of the course instances are evenly distributed with probabilities of {15, 25, 35}, {40, 50, 60}, and {70, 80, 90, 100}, respectively. This results in an average of slightly over 5,400 seats, which means that almost every student will likely have a spot in 6 courses.

Each student can be allocated up to 6 courses.

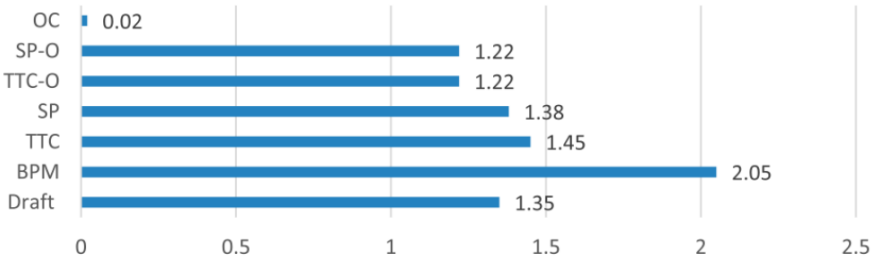
The results:

Figure 1. (Color online) The Average Number of Missed Assignment Opportunities per Algorithm



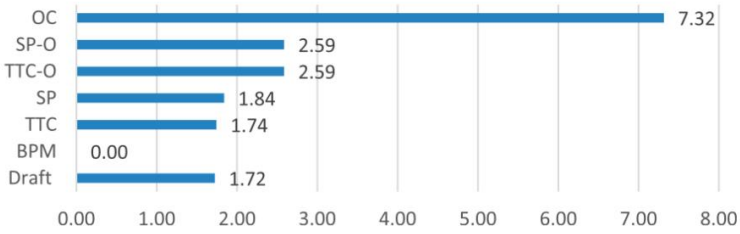
The graph shows the average misses of each algorithm. How many places out of the total number of places that the students were supposed to get (number of students * k) and didn't get

Figure 2. (Color online) The Average Range of Binary Utility per Student per Algorithm



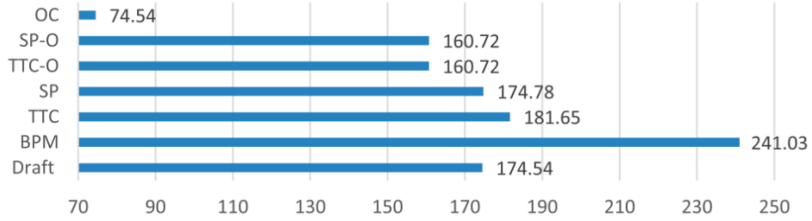
Average variance of the number of courses received by each student

Figure 3. (Color online) The Average Ordinal Utility per Student per Algorithm, Showing Amounts Above BPM Benchmark with Average Ordinal Utility of 548.15 per Student



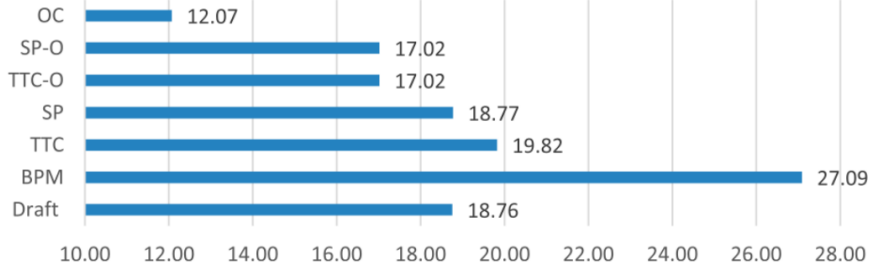
Average course ranks the students received (ordinal utility)

Figure 4. (Color online) The Average Range of Ordinal Utility per Student per Algorithm



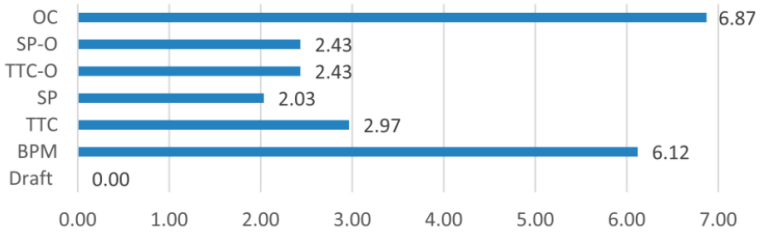
The average range between the student with the lowest ordinal utility and the student with the highest ordinal utility. The lower this average, the fairer we can consider the algorithm.

Figure 5. (Color online) Average Standard Deviation of Ordinal Utility per Student per Algorithm



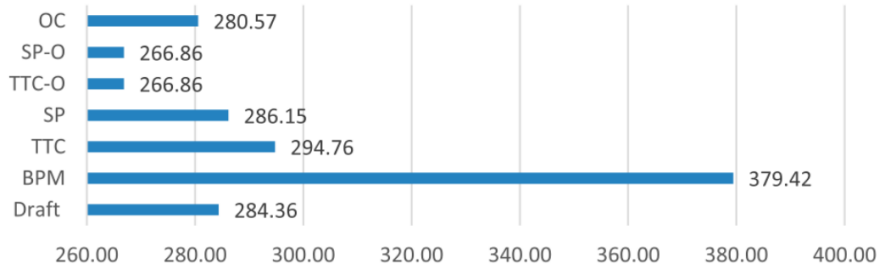
The average of the standard deviations of ordinal utility per algorithm.

Figure 6. (Color online) The Average Cardinal Utility per Algorithm, Amounts Above Draft Benchmark with Average Cardinal Utility 295.93



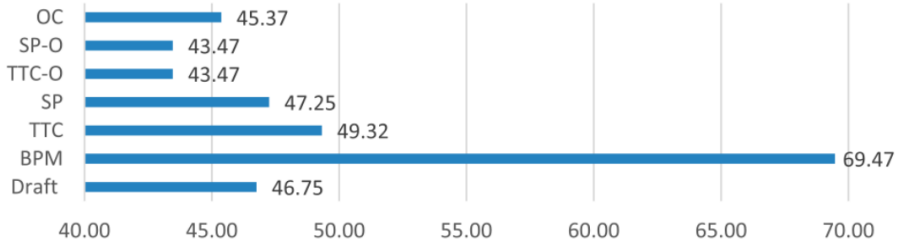
The average of the total cardinal utility sums (total bids). The higher this average, the more likely the student received the courses they preferred (since they allocated more points to them). In this figure, OC and BPM are the leading algorithms.

Figure 7. (Color online) The Average Range of Cardinal Utility per Algorithm



The average range between the student with the lowest cardinal utility and the student with the highest cardinal utility. The lower this average, the fairer we can consider the algorithm. This helps explain how BPM leads in Figure 6—by taking very high and very low cardinal sums, the average looks good, but the fairness is quite poor.

Figure 8. (Color online) Average Standard Deviation of Cardinal Utility per Algorithm



The average of the standard deviations of cardinal utility per algorithm.

Summary:

This article provides an in-depth analysis of various algorithms used for automated course allocation in universities, comparing five different algorithms: TTC (Top Trading Cycle), SP (Second Price) and OC (Ordinal-then-Cardinal). The analysis focuses on the efficiency and fairness of each algorithm. Additionally, the study compares these algorithms with two well-known methods: Draft and BPM, using metrics for fairness and efficiency. These metrics consider the total (or average), range, and standard deviation of binary, ordinal, and cardinal utility.

- Efficiency: The OC algorithm leads, with TTC and SP trailing behind.
- Fairness: The OC algorithm excels, particularly in binary and serial fairness.

Conclusion: The least effective algorithm is BPM. Among the remaining algorithms, OC stands out as the best overall.

Running Examples

In this section, we present a series of manually examples that we completed before implementing the code. These examples served as a foundational step, helping us to understand the problem better and guiding the development of our algorithms. (eventually serving as tests for our code).

Here we present 2 examples. For more go to: https://github.com/Final-Project-fairpyx/documents/tree/main/examples_for_tests

Example from the article

4 students – {s1,s2,s3,s4}

5 courses – {c1,c2,c3,c4,c5}

Available places in courses: $\{q_{c_1} = 2, q_{c_2} = 3, q_{c_3} = 3, q_{c_4} = 2, q_{c_5} = 2\}$

Each student needs 3 courses in total k=3

Total bids - 1000

Overlaps: **C1 overlaps with C4**

Student bids offers:

S1		**S2**		**S3**		**S4**	
c1	400	c3	256	c4	245	c1	251
c3	230	c2	252	c1	243	c3	242
c4	200	c4	246	c3	240	c2	235
c2	150	c1	245	c2	230	c4	201
c5	20	c5	1	c5	42	c5	71

TTC:

First round:

Voting :

c1 ($q_{1c_1} = 2$)		**c2** ($q_{1c_2} = 3$)	**c3** ($q_{1c_3} = 3$)		**c4** ($q_{1c_4} = 2$)		**c5** ($q_{1c_5} = 2$)
s1	400	-	s2	256	s3	245	
s4	251		-		-		

In course C1 there are 2 places and 2 students voting on it – S1 and S4.
Therefore both will receive C1. Note that the places in C1 runs out.
Courses C2 and C5 are not voted on at all.
In course C3 there are 3 places and only one student-S2 voted to it so he will receive C3.
In course C4 there are 2 places and only one student-S3 voted to it, therefore he will receive course C4.
In total, each student received one course and can continue to the next round.

Student bids offers:

Courses the student received
Courses dropped from availability

S1		**S2**		**S3**		**S4**	
c1	400	c3	256	c4	245	c1	251
c3	230	c2	252	c1	243	c3	242
c4	200	c4	246	c3	240	c2	235
c2	150	c1	245	c2	230	c4	201
c5	20	c5	1	c5	42	c5	71

*C1 runs out of space, C1 and C4 overlap

Second round:

Voting:

c1 ($q_{2c_1} = 0$)	**c2** ($q_{2c_2} = 3$)		**c3** ($q_{2c_3} = 2$)		**c4** ($q_{2c_4} = 1$)	**c5** ($q_{2c_5} = 2$)
-	s2	252	s4	242	-	-
	-		s3	240		
			s1	230		

In course C2 there are 3 places and one voter – S2, so S2 will receive C2.
No one voted to C4 and C5.
In course C3 there are only 2 places and three voters – S1, S2, S4. So S2 and S4, who offered the highest amount, will receive C3.

*C3 runs out of space

S1 is rejected and will therefore go through a sub-round alone

sub-round:

c1 ($q_{2c_1} = 0$)	**c2** ($q_{2c_2} = 2$)		**c3** ($q_{2c_3} = 0$)	**c4** ($q_{2c_4} = 1$)	**c5** ($q_{2c_5} = 2$)
-	s1	150	-	-	-

In course C2 there are 2 places and one voter – S1, so S1 will receive C2.

*All students received a course and can continue to the next round.

Student bids offers:

Courses the student received
Courses dropped from availability

S1		**S2**		**S3**		**S4**	
c1	400	c3	256	c4	245	c1	251
c3	230	c2	252	c1	243	c3	242
c4	200	c4	246	c3	240	c2	235
c2	150	c1	245	c2	230	c4	201
c5	20	c5	1	c5	42	c5	71

*C1 and C3 runs out of space, C1 and C4 overlap

third round:

Voting:

c1 ($q_{3c_1} = 0$)	**c2** ($q_{3c_2} = 1$)		**c3** ($q_{3c_3} = 0$)	**c4** ($q_{3c_4} = 1$)		**c5** ($q_{3c_5} = 2$)	
-	s4	235	-	s2	246	s1	20
	s3	230		-		-	

In course C4 there is 1 place and one voter – S2, so S2 will receive C4. Note that the places in C4 runs out.
In course C5 there are 2 places and one voter – S1, so S1 will receive C5.
In course C2 there is only 1 place and two voters – S3, S4. So S4, who offered the highest amount, will receive C3.

S3 is rejected and will therefore go through a sub-round alone.

sub-round:

c1 ($q_{3c_1} = 0$)	**c2** ($q_{3c_2} = 0$)	**c3** ($q_{3c_3} = 0$)	**c4** ($q_{3c_4} = 0$)	**c5** ($q_{3c_5} = 1$)	
-	-	-	-	s3	1

In course C5 there is 1 place and one voter – S3, so S3 will receive C5.

*All students have received 3 courses as they need and the algorithm ends.

Summary of TTC:

Courses the student received

Courses dropped from availability

S1		**S2**		**S3**		**S4**	
c1	400	c3	256	c4	245	c1	251
c3	230	c2	252	c1	243	c3	242
c4	200	c4	246	c3	240	c2	235
c2	150	c1	245	c2	230	c4	201
c5	20	c5	1	c5	42	c5	71

	S1	**S2**	**S3**	**S4**
Cardinal utility	570	754	527	728
Ordinal utility	8	12	9	12
Binary utility	3	3	3	3

	standard deviation	**range**	**total**
Cardinal utility	**113.017**	**227**	**2579**
Ordinal utility	**2.062**	**4**	**41**
Binary utility	**0**	**0**	**12**

s_{s_1} = {c1, c2, c5}

s_{s2} = {c3, c2, c4}

s_{s3} = {c4, c3, c5}

s_{s4} = {c1, c3, c2}

SP:

Student bids offers:

S1		**S2**		**S3**		**S4**	
c1	400	c3	256	c4	245	c1	251
c3	230	c2	252	c1	243	c3	242
c4	200	c4	246	c3	240	c2	235
c2	150	c1	245	c2	230	c4	201
c5	20	c5	1	c5	42	c5	71

First round:

Voting:

c1 ($q_{1c_1} = 2$)		**c2** ($q_{1c_2} = 3$)	**c3** ($q_{1c_3} = 3$)		**c4** ($q_{1c_4} = 2$)		**c5** ($q_{1c_5} = 2$)
s1	400	-	s2	256	s3	245	
s4	251		-		-		

In course C1 there are 2 places and two voters – S1 and S4. so S1 and S4 will receive C1. Since no one is rejected, the course will be free. Note that the places in C1 runs out.
No one voted to C2 and C5.
In course C3 there are 3 places and one voter – S2, so S2 will receive C3 for free.
In course C4 there are 2 places and one voter – S3, so S3 will receive C4 for free.

*All students received a course and can continue to the next round. All the bids are transferred to the next course of each student

Student bids offers:

Courses the student received
Courses dropped from availability

S1		**S2**		**S3**		**S4**	
c1	400	c3	256	c4	245	c1	251
c3	230	c2	252	c1	243	c3	242
c4	200	c4	246	c3	240	c2	235
c2	150	c1	245	c2	230	c4	201
c5	20	c5	1	c5	42	c5	71

*C1 runs out of space, C1 and C4 overlap

Second round:

Voting:

c1 ($q_{2c_1} = 0$)	**c2** ($q_{2c_2} = 3$)		**c3** ($q_{2c_3} = 2$)		**c4** ($q_{2c_4} = 1$)	**c5** ($q_{2c_5} = 2$)
-	s2	508	S3	728	-	-
	-		S1	630		
			S4	493		

(The bids don't match the table of price offers because the price are transferred to the next course).
In course C2 there are 3 places and one voter – S2, so S2 will receive C2 for free. No one voted to C2 and C5.
In course C3 there are 2 places and three voters – S1, S3, S4. So S1 and S3, who offered the highest amount, will receive C3. Since S4 is rejected, the price of C3 will be 493. Note that the places in C1 runs out.

S4 is rejected and will therefore go through a sub-round alone.

sub-round:

c1 ($q_{2c_1} = 0$)	**c2** ($q_{2c_2} = 2$)		**c3** ($q_{2c_3} = 0$)	**c4** ($q_{2c_4} = 1$)	**c5** ($q_{2c_5} = 2$)
-	S4	728	-	-	-

In course C2 there is 1 place and one voter – S4, so S4 will receive C2 for free.

*All students received a course and can continue to the next round.

Student bids offers:

Courses the student received
Courses dropped from availability

S1		**S2**		**S3**		**S4**	
c1	400	c3	256	c4	245	c1	251
c3	230	c2	252	c1	243	c3	242
c4	200	c4	246	c3	240	c2	235
c2	150	c1	245	c2	230	c4	201
c5	20	c5	1	c5	42	c5	71

*C1 and C3 runs out of space, C1 and C4 overlap

$P_{c_3} = 493$

Third round:

Voting:

c1 ($q_{3c_1} = 0$)	**c2** ($q_{3c_2} = 1$)		**c3** ($q_{3c_3} = 0$)	**c4** ($q_{3c_4} = 1$)		**c5** ($q_{3c_5} = 2$)	
-	S1	485	-	s2	754	S4	1000
	S3	463		-		-	

In course C4 there is 1 place and one voter – S2, so S2 will receive C4 for free.
Note that the places in C4 runs out.
In course C5 there are 2 places and one voter – S4, so S4 will receive C5 for free.
In course C2 there is only 1 place and two voters – S1, S3. So S1, who offered the highest amount, will receive C2. Since S3 is rejected, the price of C2 will be 463.

S3 is rejected and will therefore go through a sub-round alone.

sub-round:

c1 ($q_{3c_1} = 0$)	**c2** ($q_{3c_2} = 0$)	**c3** ($q_{3c_3} = 0$)	**c4** ($q_{3c_4} = 0$)	**c5** ($q_{3c_5} = 1$)	
-	-	-	-	S3	505

In course C5 there is 1 place and one voter – S41, so S1 will receive C5 for free.

*All students have received 3 courses as they need and the algorithm ends.

Summary of SP:

Courses the student received
Courses dropped from availability

S1		**S2**		**S3**		**S4**	
c1	400	c3	256	c4	245	c1	251
c3	230	c2	252	c1	243	c3	242
c4	200	c4	246	c3	240	c2	235
c2	150	c1	245	c2	230	c4	201
c5	20	c5	1	c5	42	c5	71

	S1	S2	S3	S4
Cardinal utility	780	754	527	557
Ordinal utility	10	12	10	9
Binary utility	3	3	3	3
	standard deviation		**range**	**total**
Cardinal utility	**130.911**		**253**	**2618**
Ordinal utility	**1.258**		**3**	**41**
Binary utility	**0**		**0**	**12**

s_{s_1} = {c1, c3, c5}

s_{s2} = {c3, c2, c4}

s_{s3} = {c4, c3, c2}

s_{s4} = {c1, c2, c5}

Example of Optimum changes the result

2 students - {s1, s2}

3 courses - {c1, c2, c3}

Available places in courses: $\{q_{c_1} = 1, q_{c_2} = 1\ , q_{c_3} = 1\}$

Each student needs 1 course in total k=1

Total bids - 100

Overlaps: There are no overlaps

Student bids offers:

S1		**S2**	
c1	50	c1	48
c2	49	c2	46
c3	1	c3	6

TTC:

First round:

Voting:

c1 $(q_{1c_1} = 1)$		**c2** $(q_{1c_2} = 1)$	**c3** $(q_{1c_3} = 1)$
s1	50	-	-
s2	48		

In course C1 there is only 1 place and two voters – S1, S2. So S1, who offered the highest amount, will receive C1. Note that the places in C4 runs out.
No one voted to C2 and C3.

S2 is rejected and will therefore go through a sub-round alone.

Sub round:

c1 $(q_{1c_1} = 0)$	**c2** $(q_{1c_2} = 1)$		**c3** $(q_{1c_3} = 1)$
-	s2	46	-
	-		

In course C2 there is only 1 place and one voter – S2. So S2 will receive C1. Note that the places in C2 runs out.

*All students have received 1 course as they needed, and the algorithm ends.

Summary of TTC:

Courses the student received
Courses dropped from availability

S1		**S2**	
c1	50	c1	48
c2	49	c2	46
c3	1	c3	6

	S1	**S2**
Cardinal utility	50	46
Ordinal utility	3	2
Binary utility	1	1

	standard deviation	**range**	**total**
Cardinal utility	**2.828**	**4**	**96**
Ordinal utility	**0.707**	**1**	**5**
Binary utility	**0**	**0**	**2**

S_{s_1} = {c1}

S_{s2} = { c2 }

SP:

Student bids offers:

S1		**S2**	
c1	50	c1	48
c2	49	c2	46
c3	1	c3	6

First round:

Voting:

c1 ($q_{1c_1} = 1$)		**c2** ($q_{1c_2} = 1$)	**c3** ($q_{1c_3} = 1$)
s1	50	-	-
s2	48		

In course C1 there is only 1 place and two voters – S1, S2. So S1, who offered the highest amount, will receive C1. Since S2 is rejected, the price of C1 will be 48.
Note that the places in C1 runs out.
No one voted to C2 and C3.

S2 is rejected and will therefore go through a sub-round alone.

Sub-round:

c1 ($q_{1c_1} = 0$)	**c2** ($q_{1c_2} = 1$)		**c3** ($q_{1c_3} = 1$)
-	s2	94	-
	-		

In course C2 there is 1 place and one voter – S2, so S2 will receive C2 for free.
Note that the places in C2 runs out.

*All students have received 1 course as they need, and the algorithm ends.

Summary of SP:

Courses the student received
Courses dropped from availability

S1		**S2**	
c1	50	c1	48
c2	49	c2	46
c3	1	c3	6

	S1	S2
Cardinal utility	50	46
Ordinal utility	3	2
Binary utility	1	1

	standard deviation	**range**	**total**
Cardinal utility	**2.828**	**4**	**96**
Ordinal utility	**0.707**	**1**	**5**
Binary utility	**0**	**0**	**2**

s_{s_1} = {c1}

s_{s2} = {c2}

Optimal (TTC-O, SP-O)

The optimal algorithms calculate in each round the maximum ordinal possible for that round and within that will choose the courses that will give the maximum possible bids.

Student bids offers:

S1		**S2**	
c1	50	c1	48
c2	49	c2	46
c3	1	c3	6

First round:

The highest possible ordinal amount is for one student to receive the first course in the offer and the other the second. (4+5=9) – because there is no place for both students in their initial proposal.

In this case, the possible **cardinal sums** are:

S1 will receive C1 & S2 will receive C2 = 96

S1 will receive C2 & S2 will receive C1 = 97

The second sum is larger, so the optimal algorithm will choose it.

*All students have received 1 course as they needed, and the algorithm ends.

Summary of the optimal algorithms:

Courses the student received
Courses dropped from availability

S1		**S2**	
c1	50	c1	48
c2	49	c2	46
c3	1	c3	6

	S1	S2
Cardinal utility	49	48
Ordinal utility	2	3
Binary utility	1	1

	standard deviation	**range**	**total**
Cardinal utility	**0.707**	**1**	**97**
Ordinal utility	**0.707**	**1**	**5**
Binary utility	**0**	**0**	**2**

s_{s_1} = {c2}

s_{s2} = {c1}

Optimal (OC)

OC calculate (not in round but in total) the maximum possible ordinal and within that will choose the courses that will give the maximum possible bids.

(In this case there is a single round so it will work same as TTC-O & SP-O)

Student bids offers:

S1		**S2**	
c1	50	c1	48
c2	49	c2	46
c3	1	c3	6

Available places in courses: $\{q_{c_1} = 1, q_{c_2} = 1\ , q_{c_3} = 1\}$

Each student needs 1 course in total k=1

In this case, the possible **cardinal sums** are:

S1 will receive C1 & S2 will receive C2 = 96

S1 will receive C2 & S2 will receive C1 = 97

The second sum is larger, so the optimal algorithm will choose it.

*All students have received 1 course as they need, and the algorithm ends.

Summary of OC:

Courses the student received
Courses dropped from availability

S1		**S2**	
c1	50	c1	48
c2	49	c2	46
c3	1	c3	6

	S1	S2
Cardinal utility	49	48
Ordinal utility	2	3
Binary utility	1	1

	standard deviation	**range**	**total**
Cardinal utility	**0.707**	**1**	**97**
Ordinal utility	**0.707**	**1**	**5**
Binary utility	**0**	**0**	**2**

s_{s_1} = {c2}

s_{s2} = {c1}

Performance comparison

In this section, we compared the performance of our five algorithms (TTC, SP, TTC-O, SP-O, OC) with two existing algorithms from the fairpyx library (iterated_maximum_matching_unadjusted, iterated_maximum_matching_adjusted) across three categories:

1. **Uniform** - Randomly generated data.
2. **Ariel** - Data provided by students from Ariel University.
3. **SZWS** - Data sourced from a research article.

   (https://arxiv.org/abs/2210.00954 , in section 5.5)

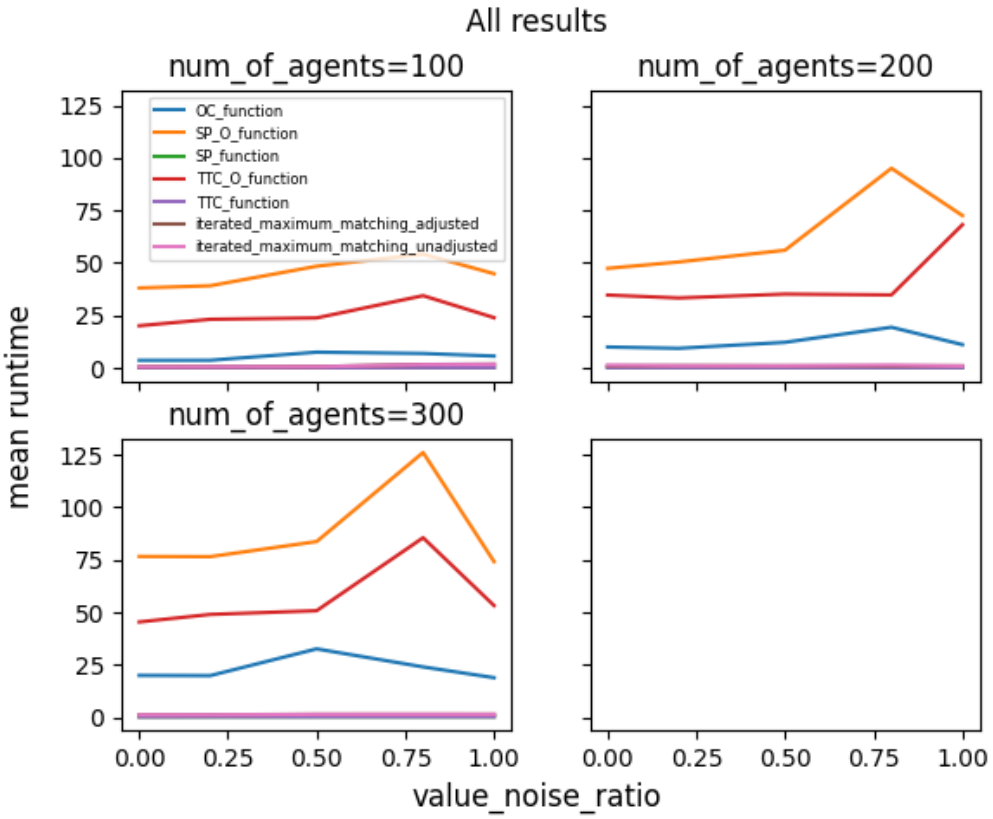
Notions:

- Value noise ratio: the ratio between the value of the courses and the bids students placed on each course—the higher the ratio, the more diverse the course preferences among students.
- Supply ratio: The ratio of available spots in a course to the number of students who want that course.
- Popular item: The number of courses that are in high demand by most students.
- Top x: The number of students who received their Xth-ranked course on their list.
- Deficit: The number of courses a student wanted but did not receive.
- Envy: The level of envy among students based on the courses they were allocated.
  Envy is calculated by comparing the value that each student places on the courses received by other students. For each student, calculate how much they would value the courses that another student received, and subtract the total value of the courses they actually received. If the result is positive, the student feels envy towards the other student because they perceive the other student's allocation as more valuable. If the result is negative, the student does not feel envy, as they value their own allocation more or equally compared to the other student's allocation.
- Egalitarian: the degree of fairness in the course allocation.
  It calculated by the average of the student with the lowest total bids
- Utilitarian: the total sum of bids for all allocated courses.

UNIFORM

1. **Runtime Plot**
   Overview: These plots show the mean runtime of all the algorithms as a function of the value noise ratio, for different numbers of agents (100, 200, and

300).

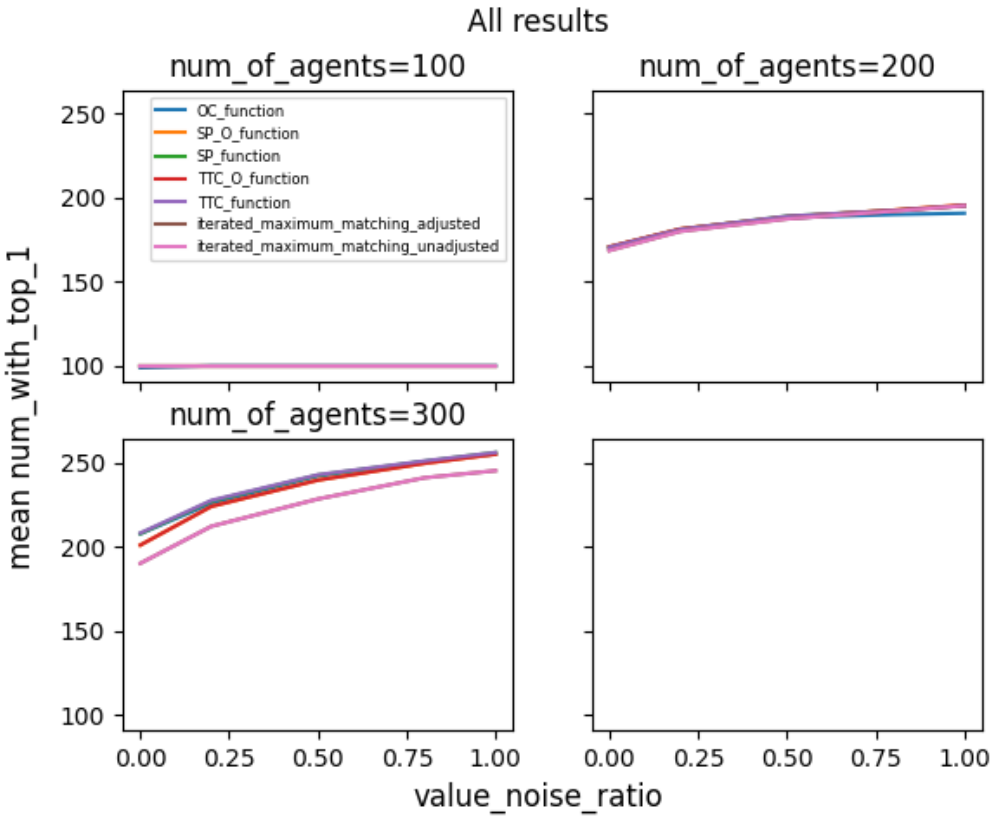


Interpretation: As the number of agents increases, the runtime generally increases for all algorithms, which is expected due to the higher computational load.

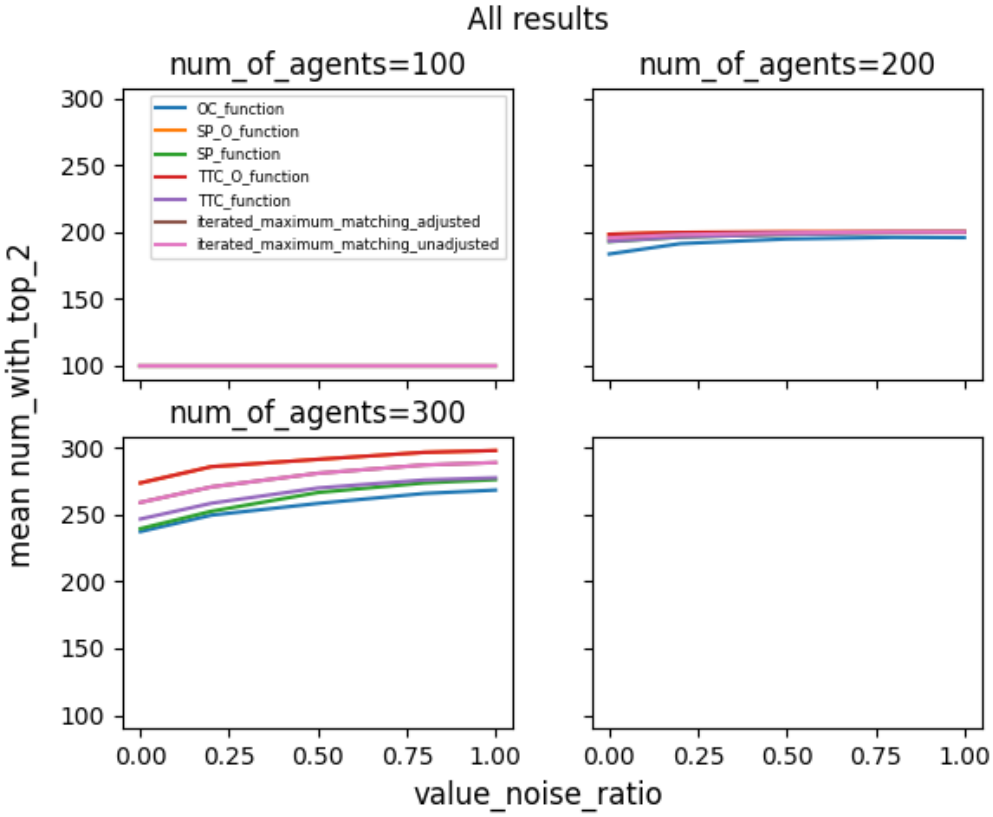
The OC_function, TTC_O_function and SP_0_function show higher runtime, especially for a larger number of agents, indicating they are more computationally intensive. Other algorithms, particularly the iterated matching algorithms, maintain lower runtimes, making them more efficient.

2. **Top 1, Top 2, and Top 3 Satisfaction**
   Overview: These plots display the number of agents who received one of their top 1, top 2, or top 3 choices, depending on the value noise ratio and the number of agents.

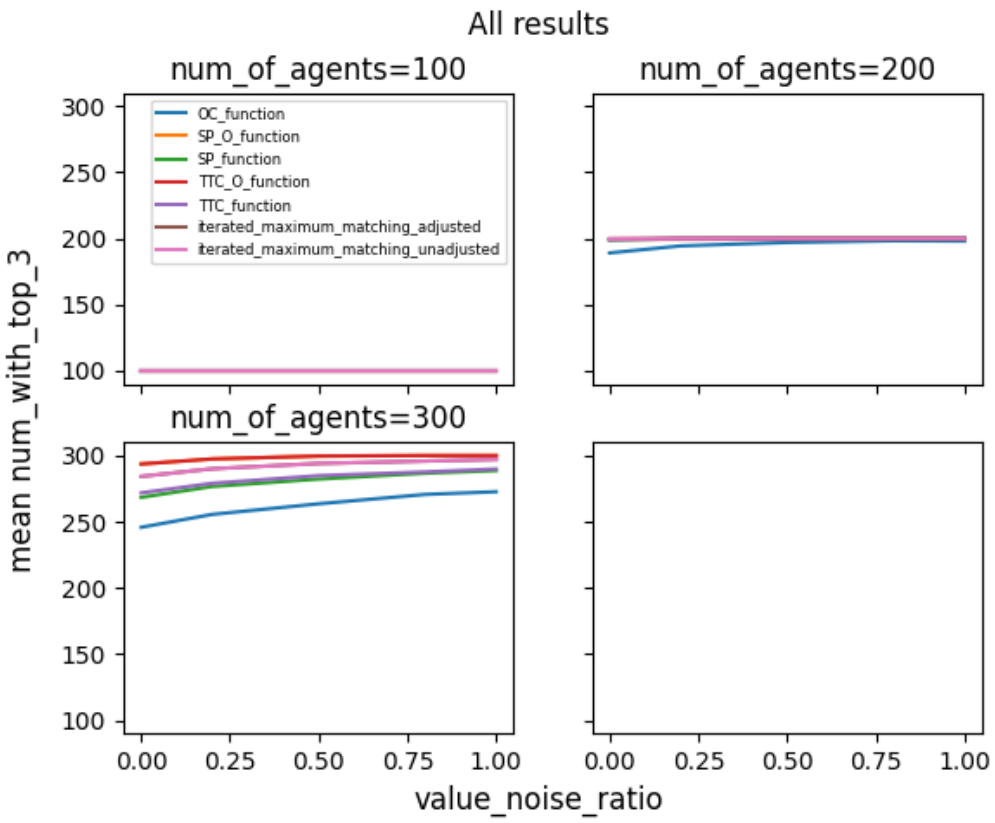
Top 1:



Top 2:



Top 3:

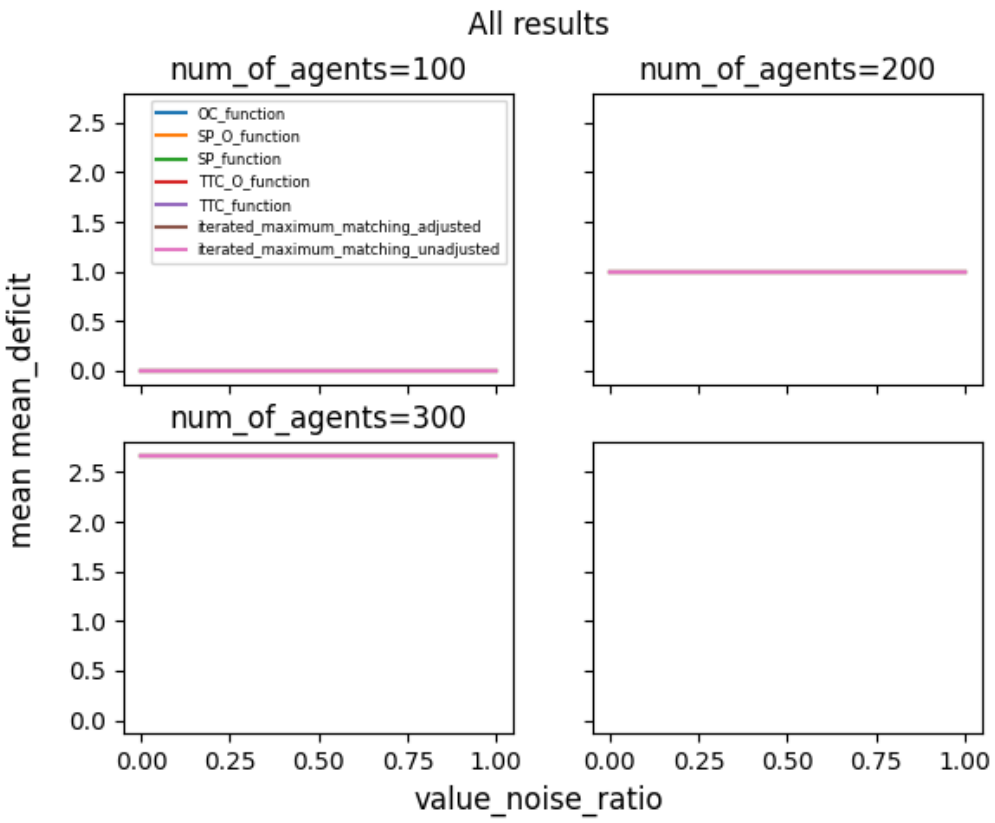


Interpretation: Across all metrics (top 1, top 2, and top 3), most algorithms perform very well, with high satisfaction rates..
There is a slight decrease in satisfaction as the number of agents increases, but the impact is minimal.
The performance of all algorithms is quite close, with only minor differences, suggesting that they are all capable of maintaining high satisfaction levels even under varying noise conditions.

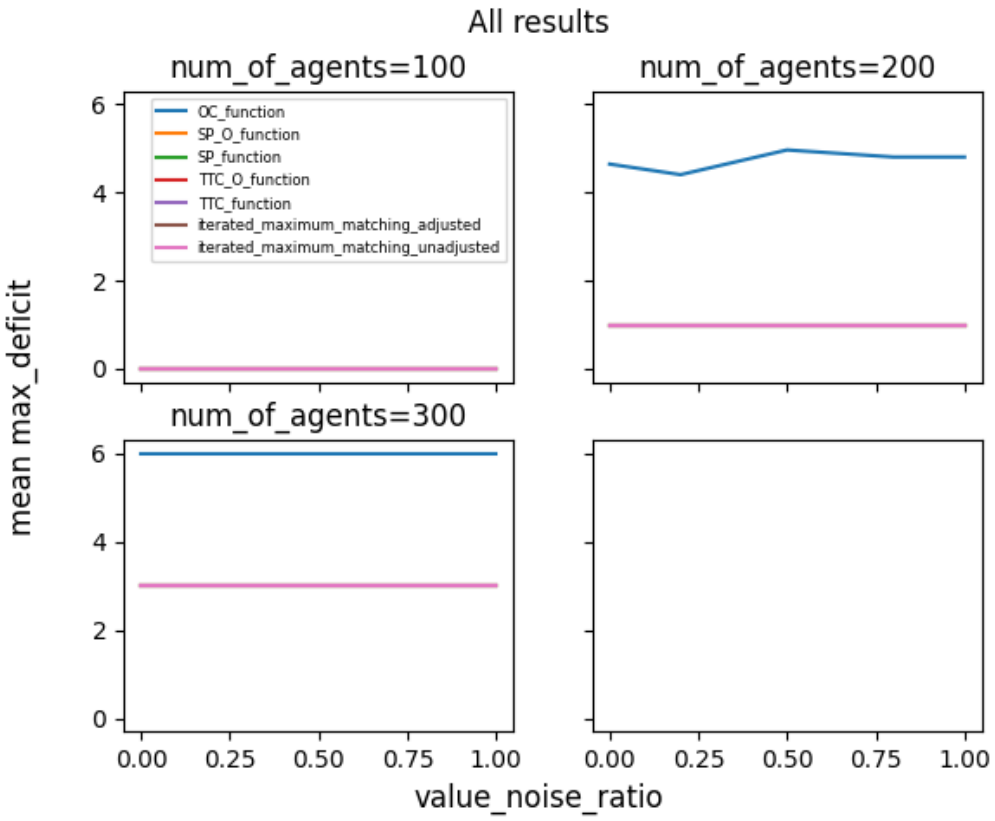
3. **Mean and Max Deficit**

Overview: These plots measure the mean and maximum deficit in the allocation process as a function of the value noise ratio.

mean deficit:



max deficit:



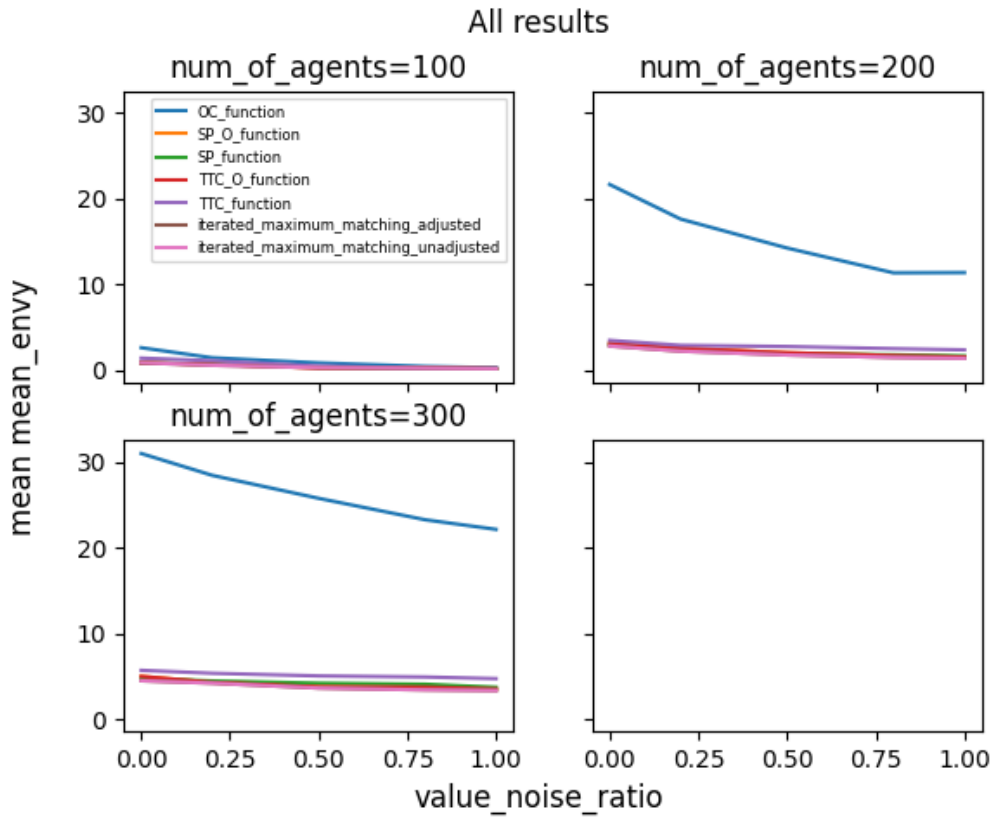
Interpretation: The iterated matching algorithms show no deficit across all

conditions, indicating they are effective in fully allocating resources.
OC_function, TTC_O_funcion and SP_0_function show some deficit, especially as the noise ratio increases, but the deficit remains relatively low.
The deficit remains consistent across different numbers of agents, indicating stable performance across varying problem sizes.

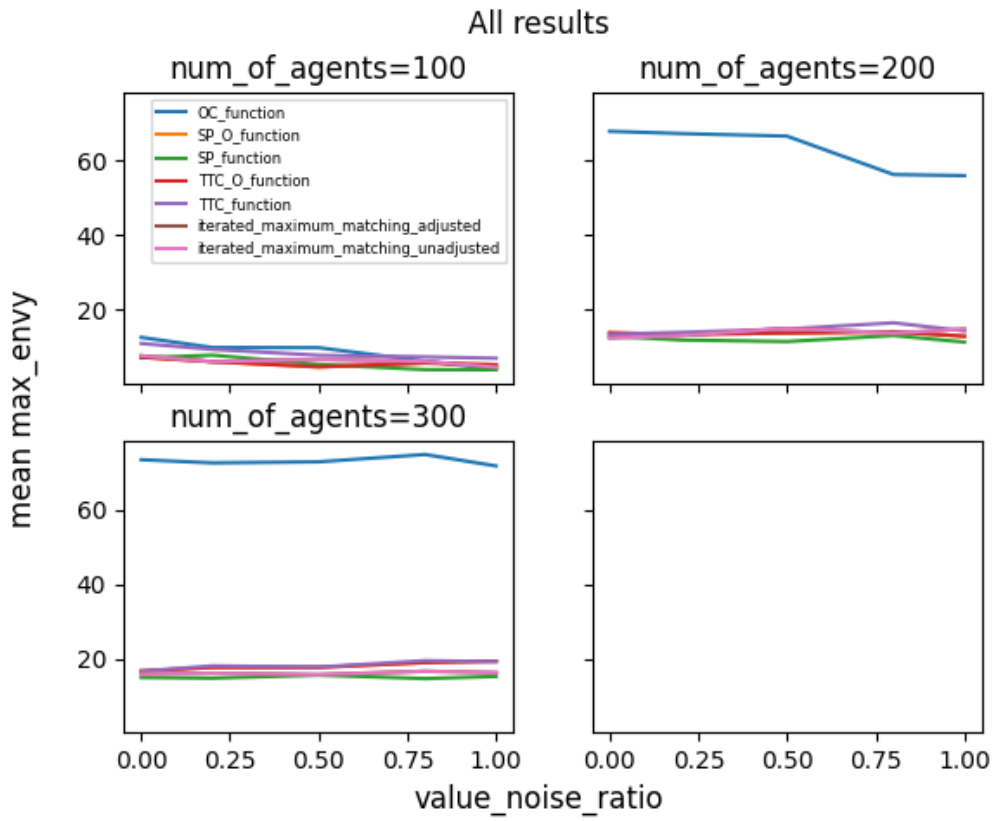
4. **Mean and Max Envy**

Overview: These plots represent the level of envy, with lower values indicating a more fair allocation.

mean envy:



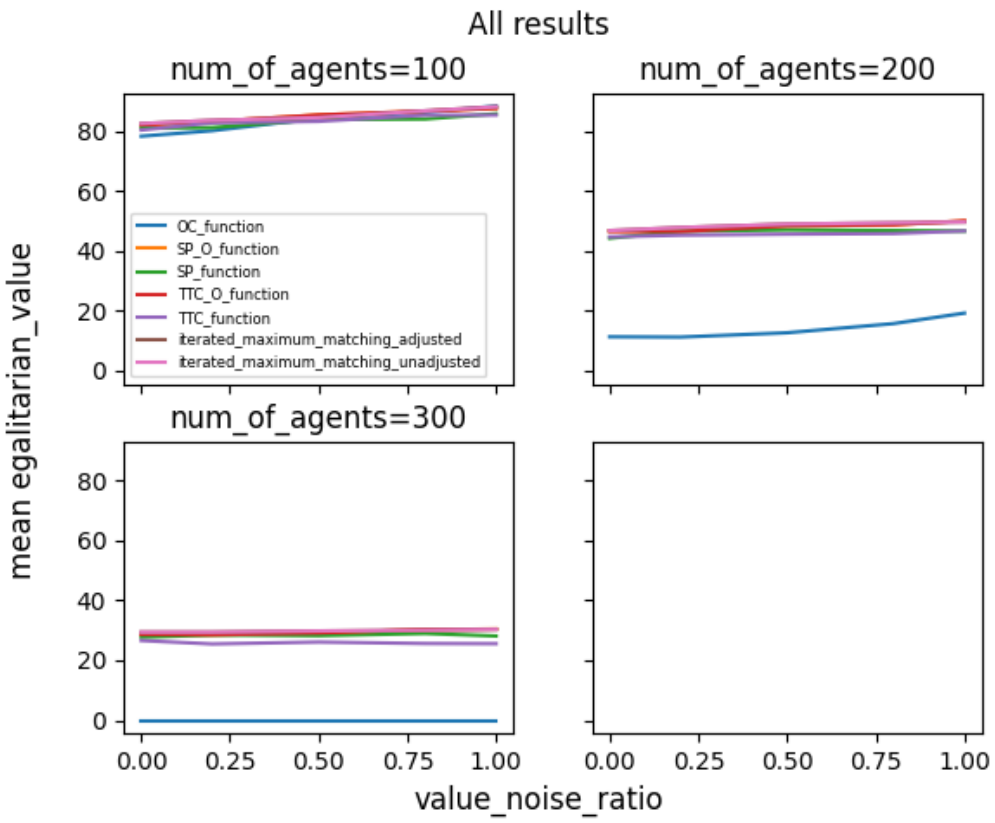
max envy:



Interpretation: The mean and max envy tend to decrease slightly as the value noise ratio increases, which suggest that increased randomness in preferences reduces envy.
OC_function consistently shows higher envy compared to other algorithms, particularly when there are fewer agents.
Iterated matching algorithms perform the best in terms of minimizing envy, especially in scenarios with a higher number of agents.

5. **Egalitarian Value**

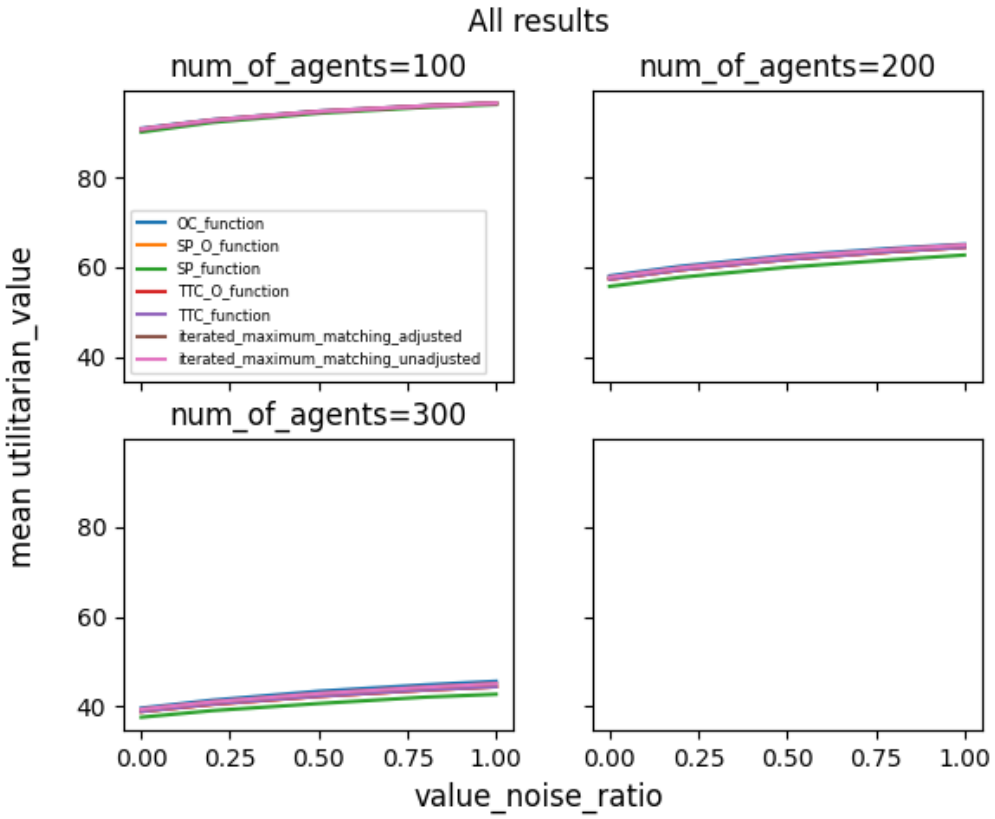
Overview: This plot shows the mean egalitarian value, representing the fairness of the allocation.



Interpretation: The egalitarian value increases slightly as the value noise ratio increases, indicating that the fairness of the allocation improves with more noise in preferences.
The iterated matching algorithms perform consistently well across all scenarios, maintaining higher egalitarian values.
OC_function lags behind in terms of fairness, particularly as the number of agents increases.

6. **Utilitarian Value**
   Overview: This plot displays the mean utilitarian value, representing overall

welfare or satisfaction.

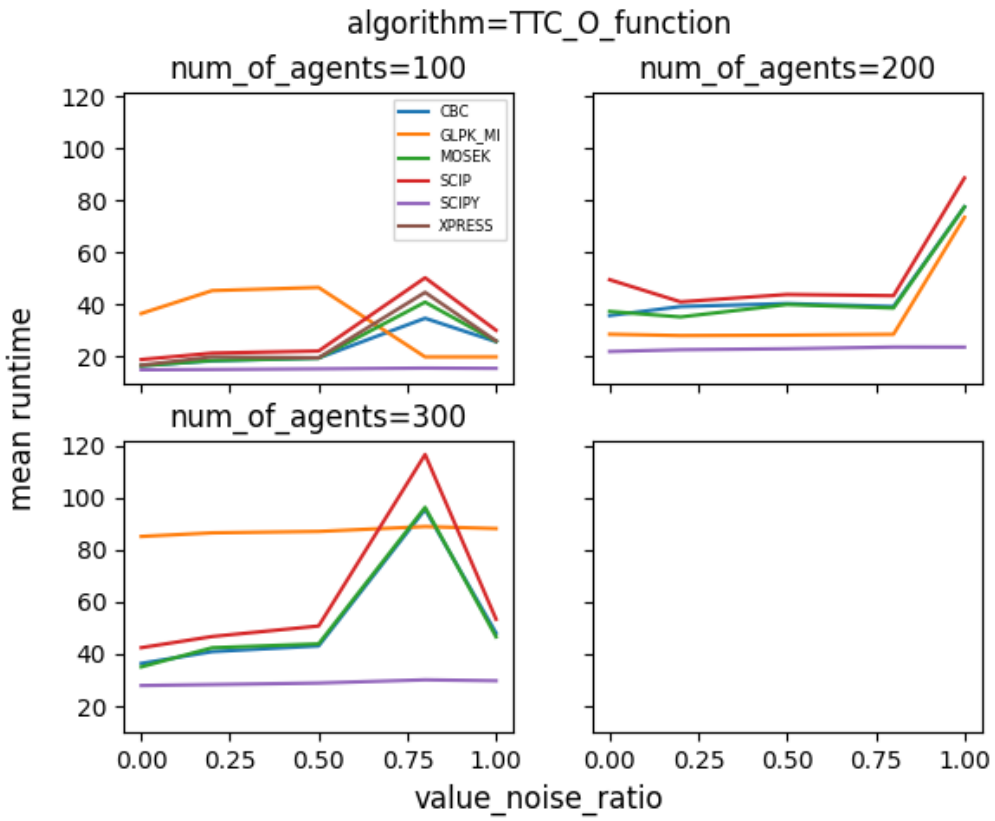


Interpretation: All algorithms show an increase in utilitarian value as the value noise ratio increases, suggesting that the overall satisfaction improves with more randomness in preferences.
Differences among algorithms are minimal, indicating that all of them are effective in maximizing welfare- the sum of the total bids.

7. **Solvers**

Overview: To enhance the performance of our algorithms, we implemented the option for users to select a solver from the cvxpy library. This plot illustrates the average runtime of each solver when applied to the optimization algorithms.
solver we used**:** CBC, GLPK_MI, MOSEK, SCIP, SCIPY and XPRESS

TTC-O:



Interpretation:

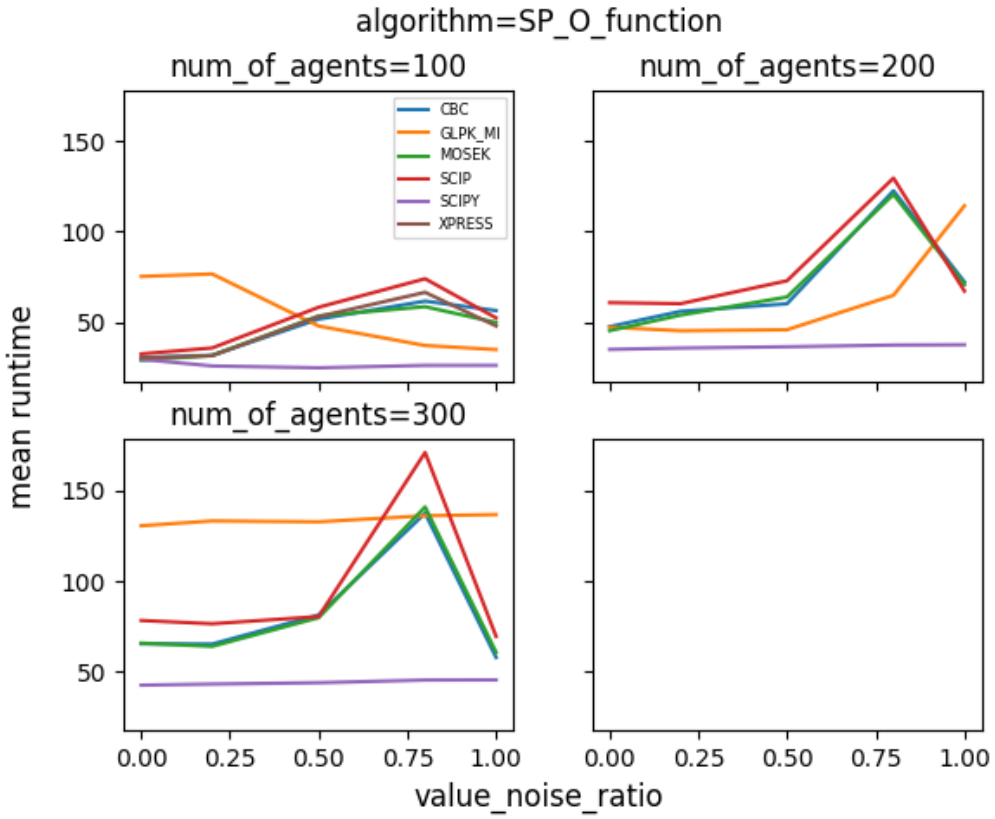
1. SCIPY continues to deliver the best performance, maintaining the lowest runtimes.

2. SCIP and MOSEK exhibit increasing runtimes, particularly at higher noise ratios and agent counts, with SCIP being the most affected.

3. GLPK_MI and CBC have moderate runtimes but show some instability at higher noise ratios, especially with 300 agents.

4. XPRESS remains stable but slower than SCIPY, especially as conditions become more complex.

SP-O:



Interpretation:

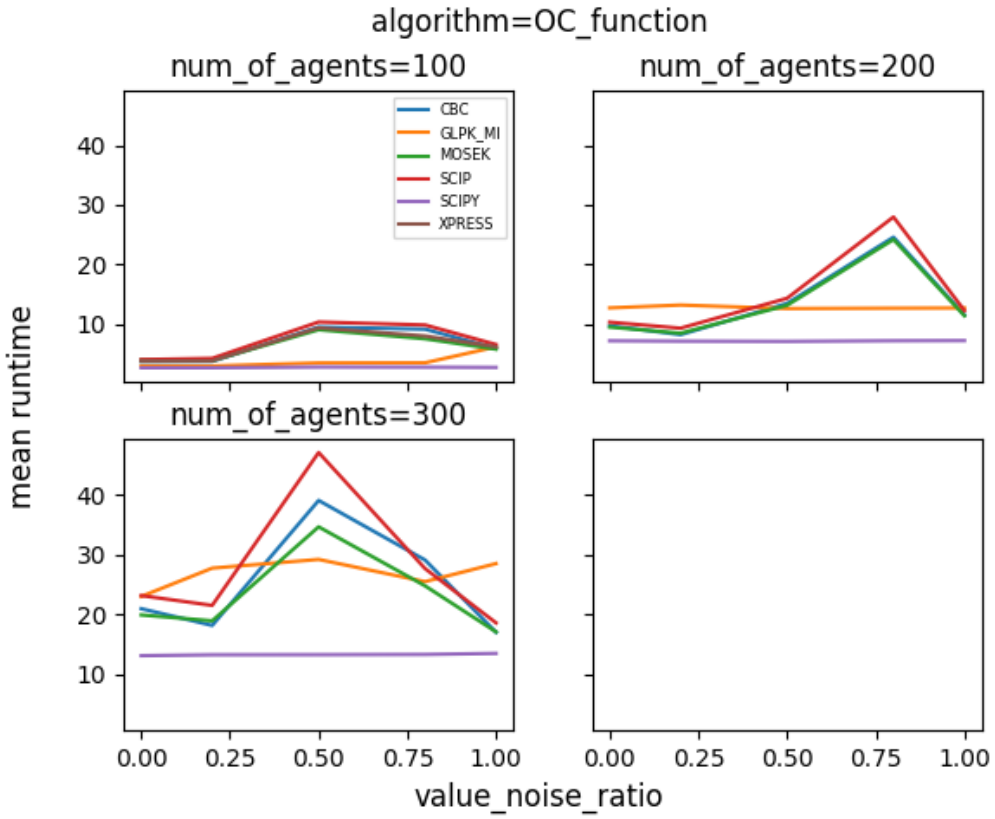
1. SCIPY again outperforms all other solvers with the lowest runtimes across different scenarios.

2. SCIP shows significant increases in runtime as the noise ratio approaches 0.75, especially with 200 and 300 agents, indicating potential inefficiency under these conditions.

3. MOSEK and GLPK_MI also exhibit increased runtimes with higher noise ratios but remain relatively stable compared to SCIP.

4. CBC shows fluctuating performance with higher runtimes, particularly at higher noise ratios and agent counts.

OC:



Interpretation:

1. SCIPY maintains the lowest runtimes across all value noise ratios and agent counts, showing consistent efficiency.

2. SCIP and MOSEK have moderate runtimes, with noticeable increases as the value noise ratio approaches 0.75, particularly with 200 and 300 agents.

3. CBC and GLPK_MI also show increasing runtimes with higher noise ratios, but they generally perform better than SCIP.

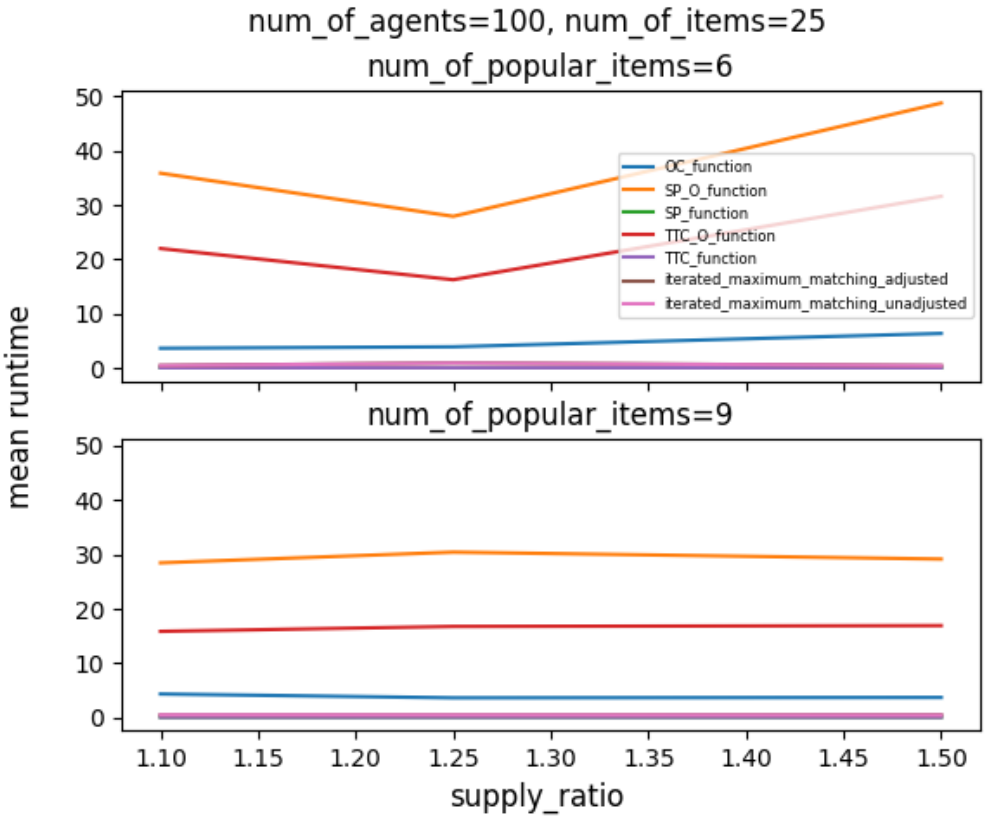
4. XPRESS is stable but generally slower than SCIPY, especially as agent count and noise ratio increase.

Summary: SCIPY remains the most efficient solver across all tested algorithms and scenarios, consistently delivering the lowest runtimes. SCIP and MOSEK show significant runtime increases with higher value noise ratios, especially when agent counts are high, making them less suitable for scenarios with significant noise. CBC and GLPK_MI provide moderate performance but are less stable under more complex conditions.

SZWS

1. **Runtime Plot**

   Overview: These plots show the mean runtime of all the algorithms as a function of supply ratio, for different numbers of popular items.

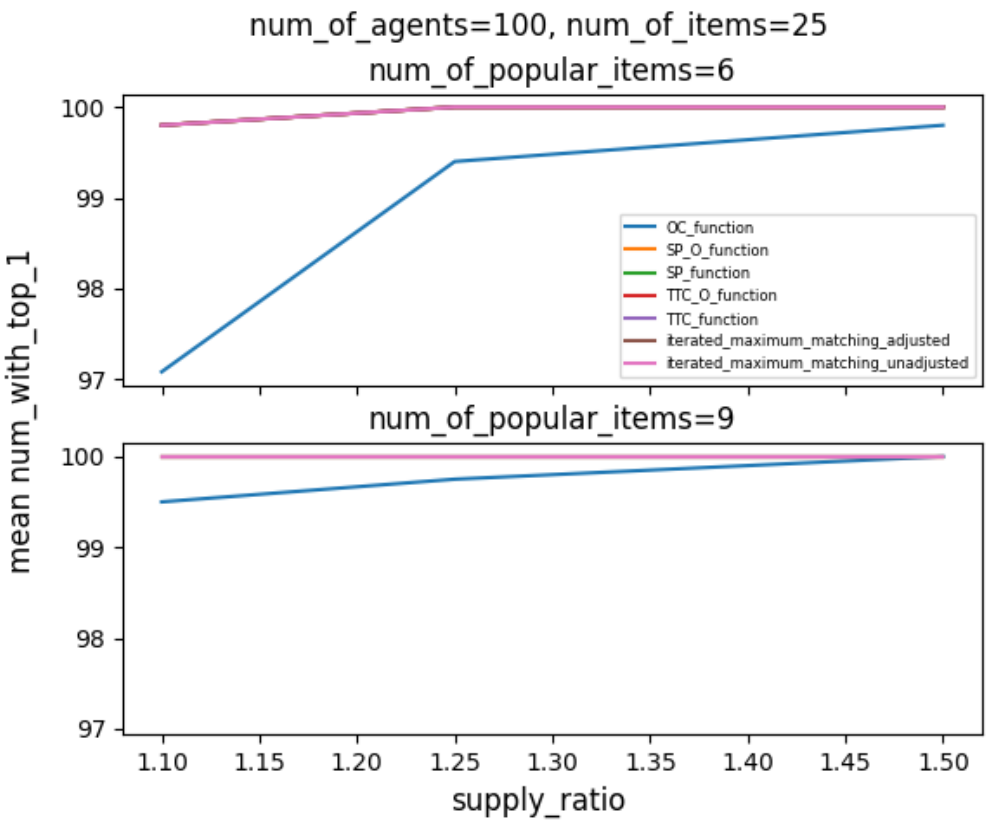


   Interpretation: The SP_O_function exhibits the highest runtime across all supply ratios, indicating that it is the most computationally intensive and least efficient, with significantly longer runtimes, particularly as the supply ratio changes. The TTC_function and OC_function have moderate runtimes with minimal fluctuation as the supply ratio varies. The Iterated_maximum_matching (both adjusted and unadjusted) algorithms consistently show the lowest and most stable runtimes across all conditions, making them the most efficient and ideal for larger or more complex scenarios due to their minimal runtime variation.

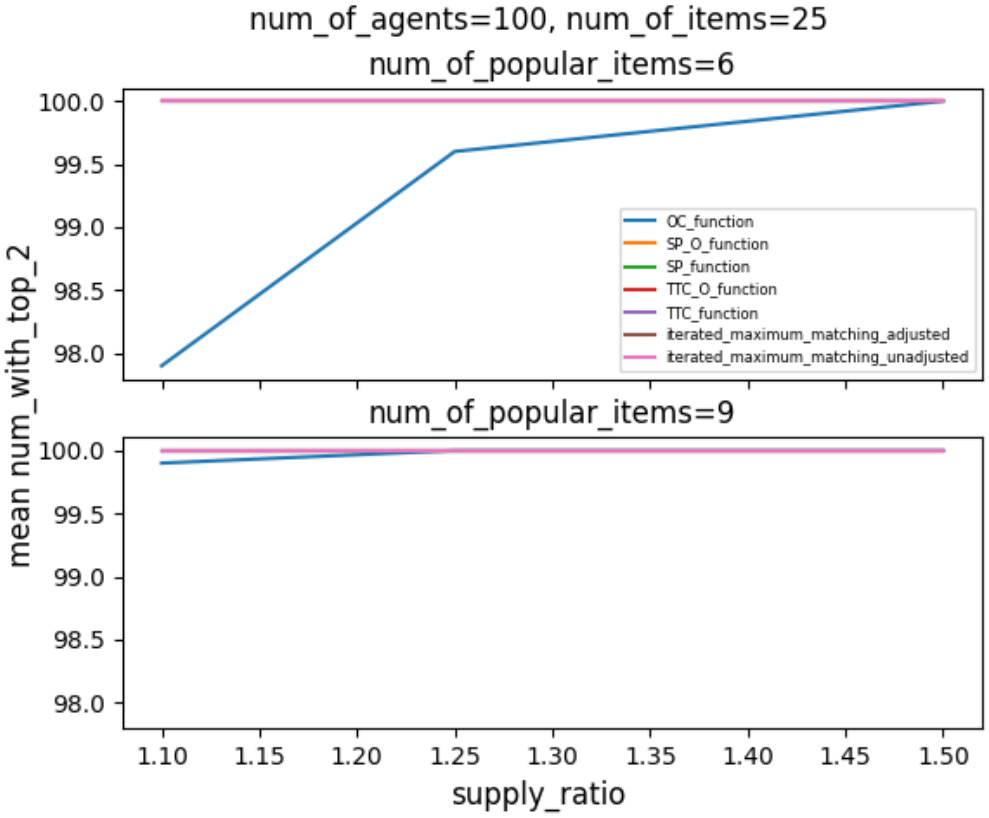
2. **Top 1, Top 2, and Top 3 Satisfaction**

   Overview: These plots display the number of agents who received one of their top 1, top 2, or top 3 choices, depending on the supply ratio and the number of popular items.

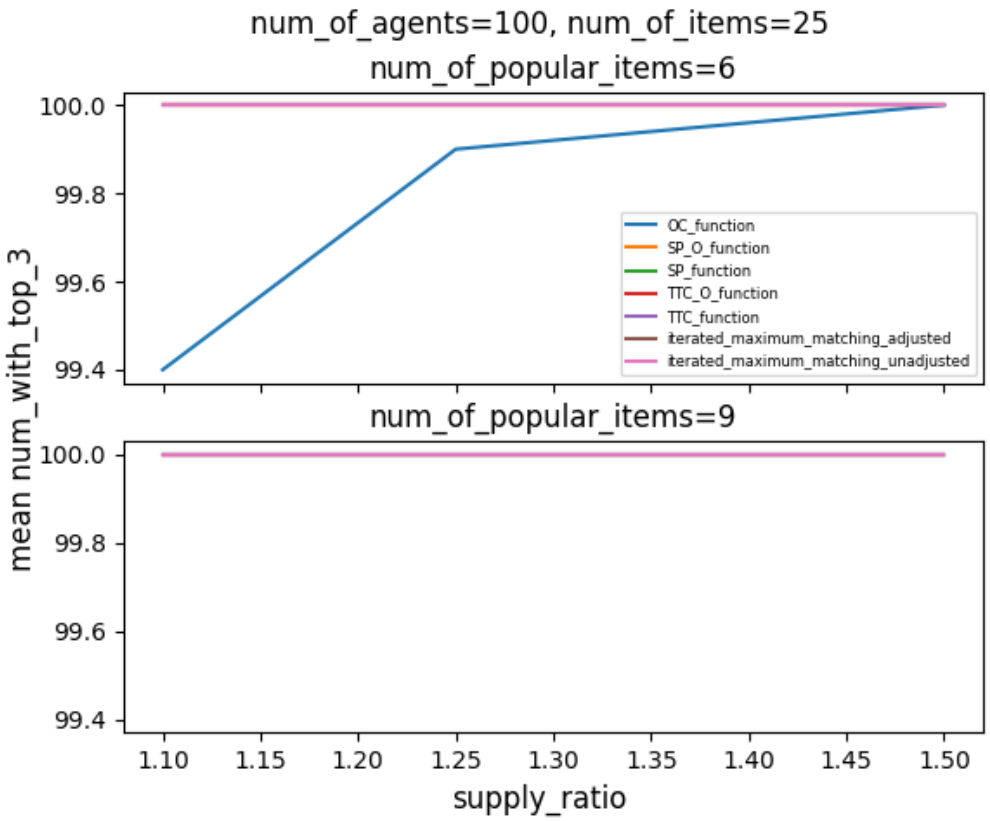
Top 1:


num_of_agents=100, num_of_items=25
num_of_popular_items=6
num_of_popular_items=9
mean num_with_top_1
supply_ratio
OC_function
SP_O_function
SP_function
TTC_O_function
TTC_function
iterated_maximum_matching_adjusted
iterated_maximum_matching_unadjusted

Top 2:


num_of_agents=100, num_of_items=25
num_of_popular_items=6
num_of_popular_items=9
mean num_with_top_2
supply_ratio
OC_function
SP_O_function
SP_function
TTC_O_function
TTC_function
iterated_maximum_matching_adjusted
iterated_maximum_matching_unadjusted

Top 3:

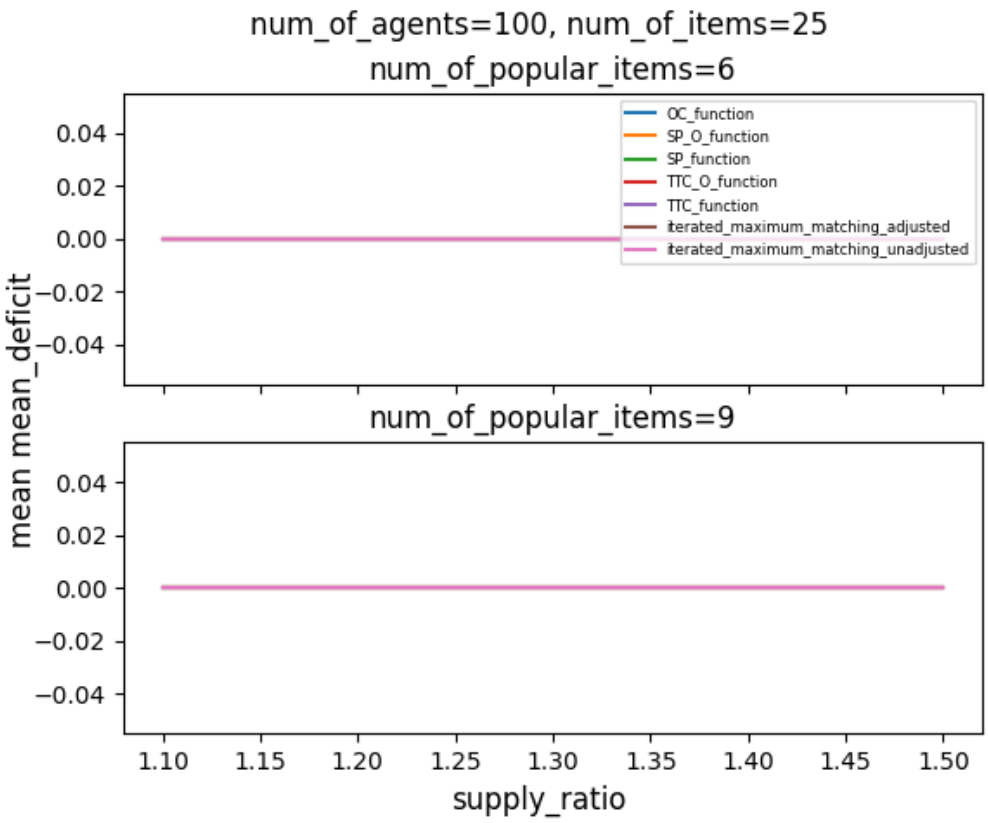


Interpretation: Across all metrics (top1, top 2, and top 3), most algorithms perform very well, with nearly all students receiving one of their top choices, especially as the supply ratio increases.
The OC_function shows a slight variation, particularly in the lower supply ratio, but catches up at higher supply ratios.
The other algorithms remain near perfect, indicating they are very effective in terms of student satisfaction.

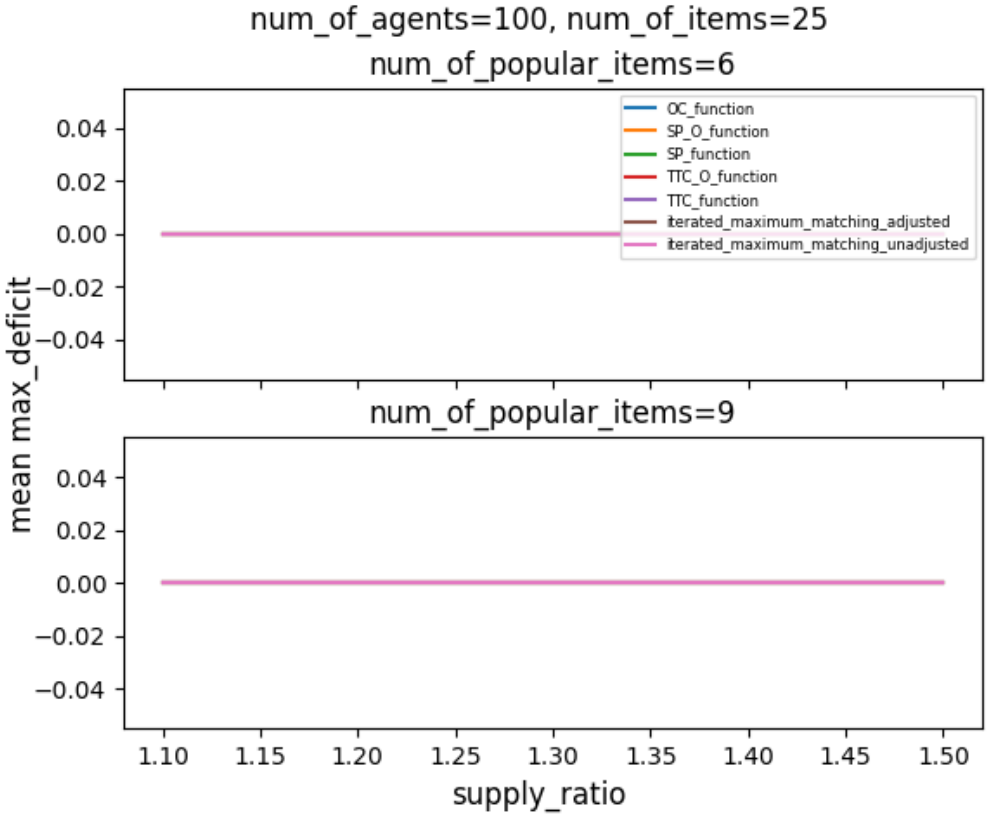
3. **Mean and Max Deficit**

Overview: These plots measure the mean and maximum deficit in the allocation process as a function of the supply ratio.

mean deficit:



max deficit:



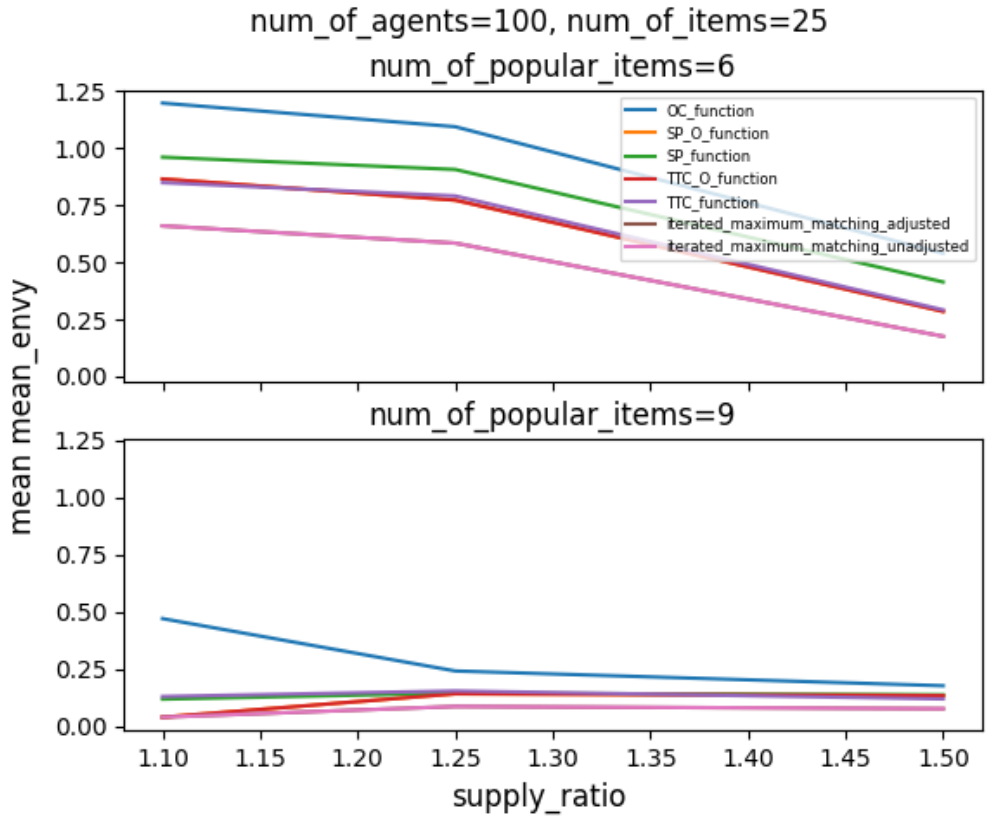
Interpretation: All algorithms show zero deficits across the board, indicating that

they effectively allocate all resources without leaving any unmet demand. This suggests strong performance in resource allocation by all algorithms, especially under the given parameters.

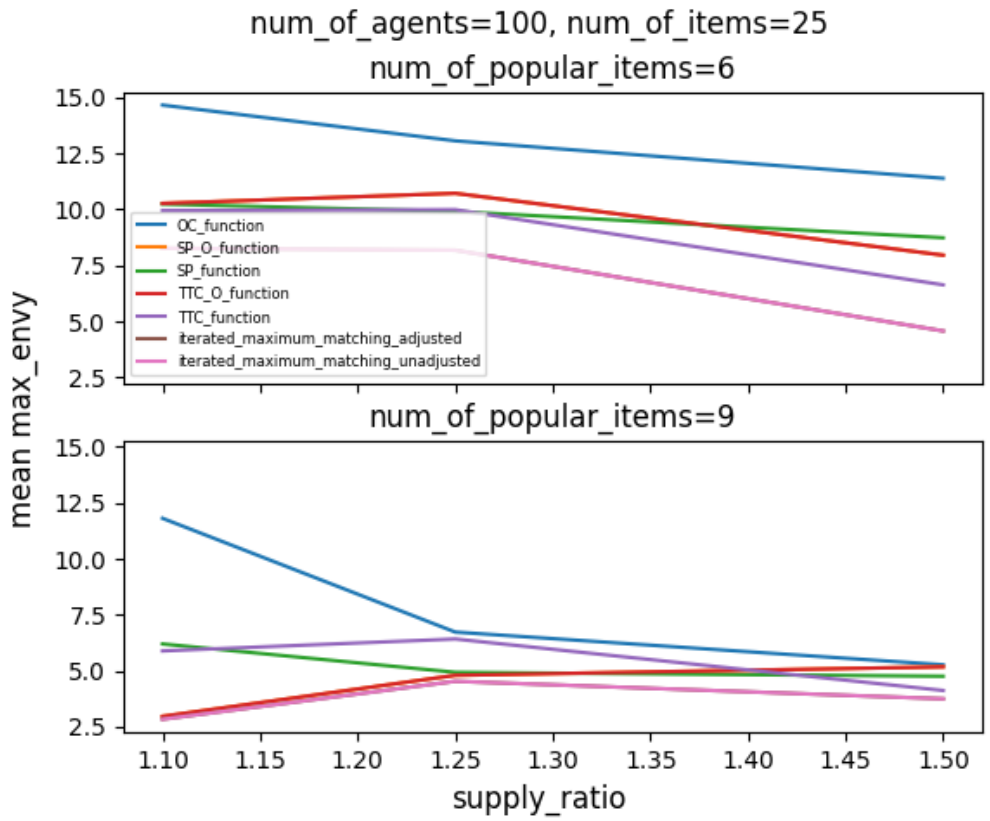
4. Mean and Max Envy

Overview: These plots represent the level of envy, with lower values indicating a more fair allocation.

mean envy:



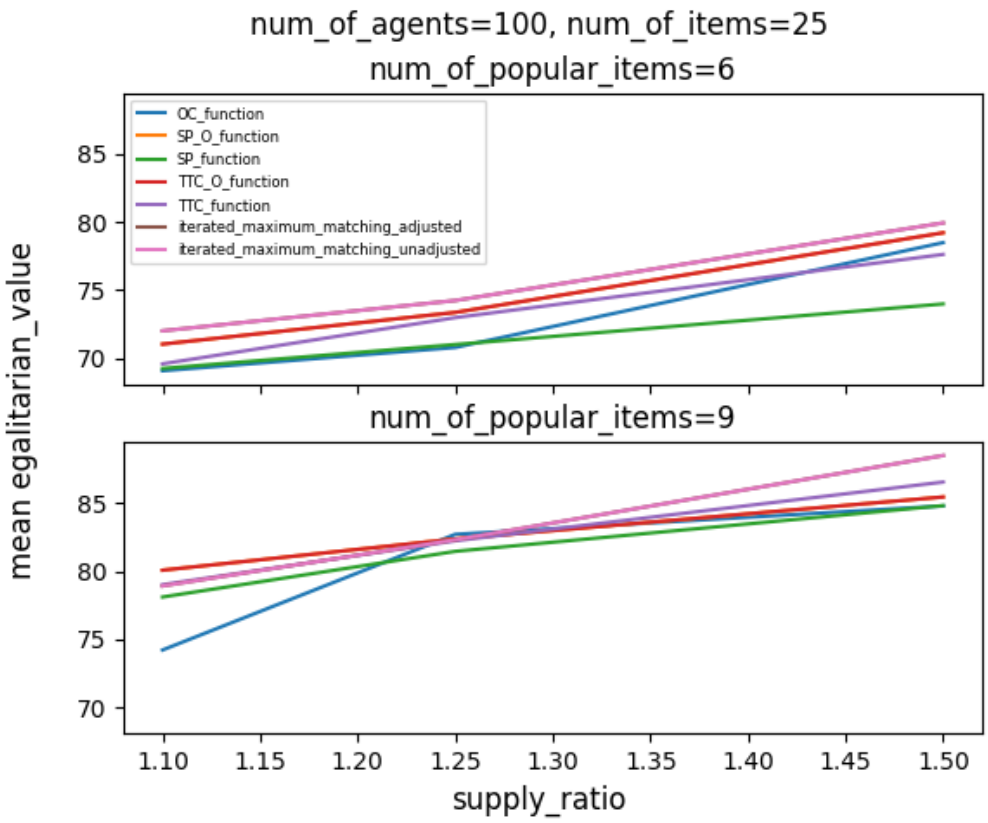
max envy:



Interpretation: Envy tends to decrease as the supply ratio increases, which makes sense since a higher supply generally means less competition for popular courses.
The OC_function seems to start with higher envy but decreases with a higher supply ratio.
Other algorithms show relatively lower envy values, particularly the iterated matching algorithms, which seem to be quite effective in reducing envy.

5. Egalitarian Value

Overview: This plot shows the mean egalitarian value, representing the fairness of the allocation.

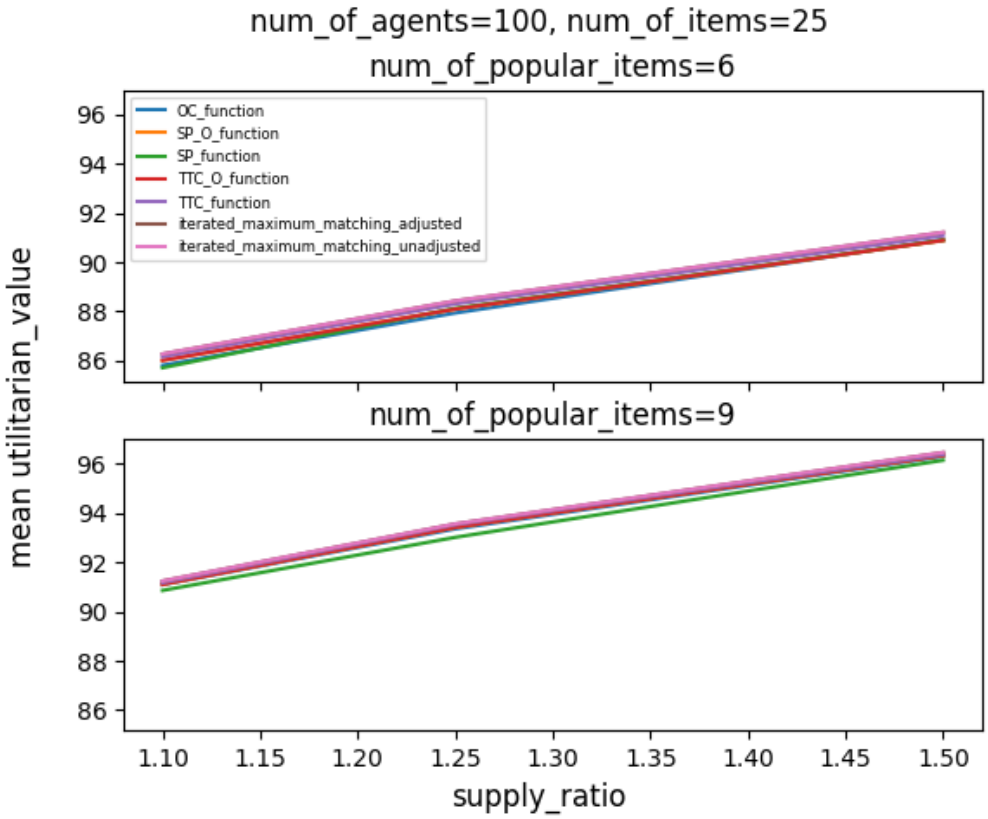


Interpretation: As the supply ratio increases, the egalitarian value increases for all algorithms.
The iterated matching algorithms, particularly the unadjusted one, seem to perform consistently well, achieving higher egalitarian values than others.
OC_function shows some variation but improves as the supply ratio increases.

6. **Utilitarian Value**

Overview: This plot displays the mean utilitarian value, representing overall

welfare or satisfaction.

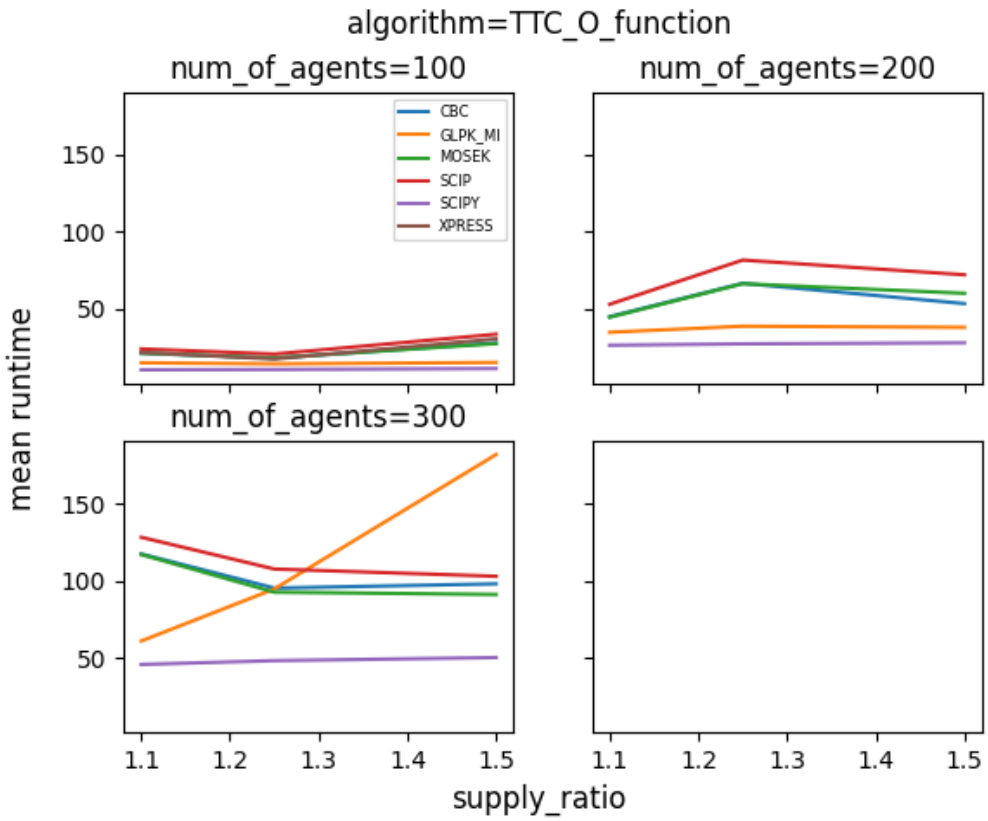


Interpretation: Utilitarian value increases with the supply ratio, which is expected since more resources lead to better overall outcomes.
All algorithms converge to similar utilitarian values at higher supply ratios, suggesting that they all perform well in maximizing overall satisfaction.

7. Solvers

Overview: To enhance the performance of our algorithms, we implemented the option for users to select a solver from the cvxpy library. This plot illustrates the average runtime of each solver when applied to the optimization algorithms.
solver we used**:** CBC, GLPK_MI, MOSEK, SCIP, SCIPY and XPRESS

TTC-O:



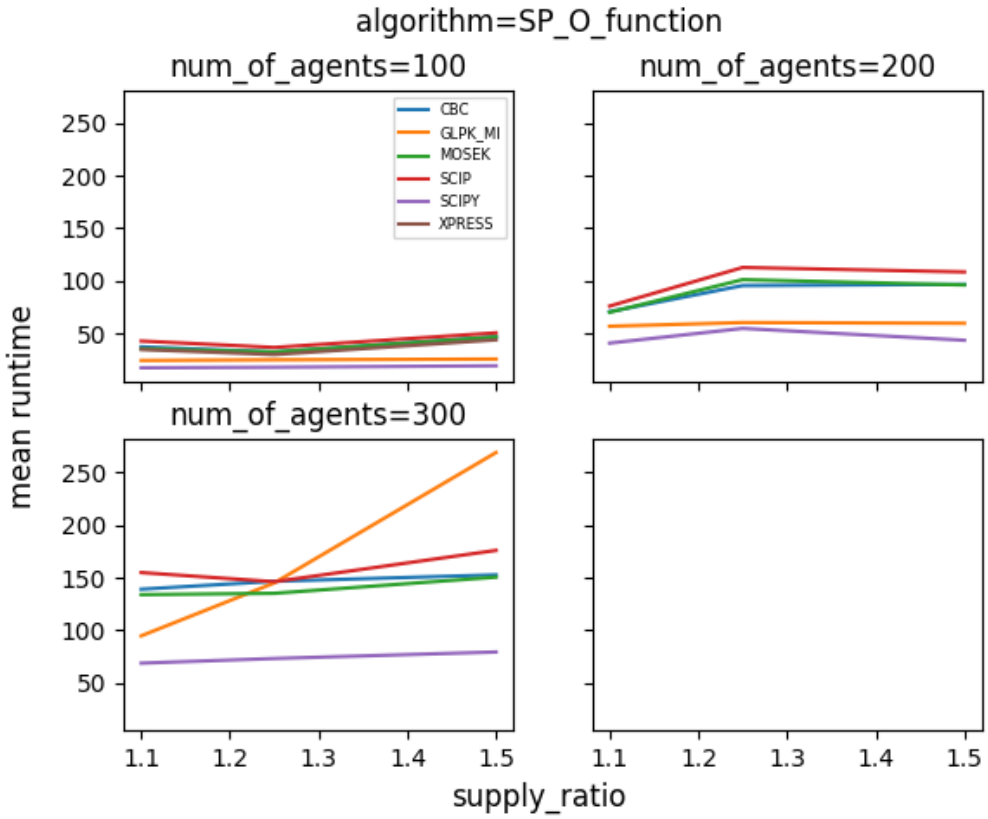
Interpretation:

1. SCIPY continues to outperform with the lowest runtimes across different conditions.

2. CBC and MOSEK show significant runtime increase with 300 agents, particularly at higher supply ratios.

3. GLPK_MI and XPRESS are stable but generally slower than SCIPY.

SP-O:



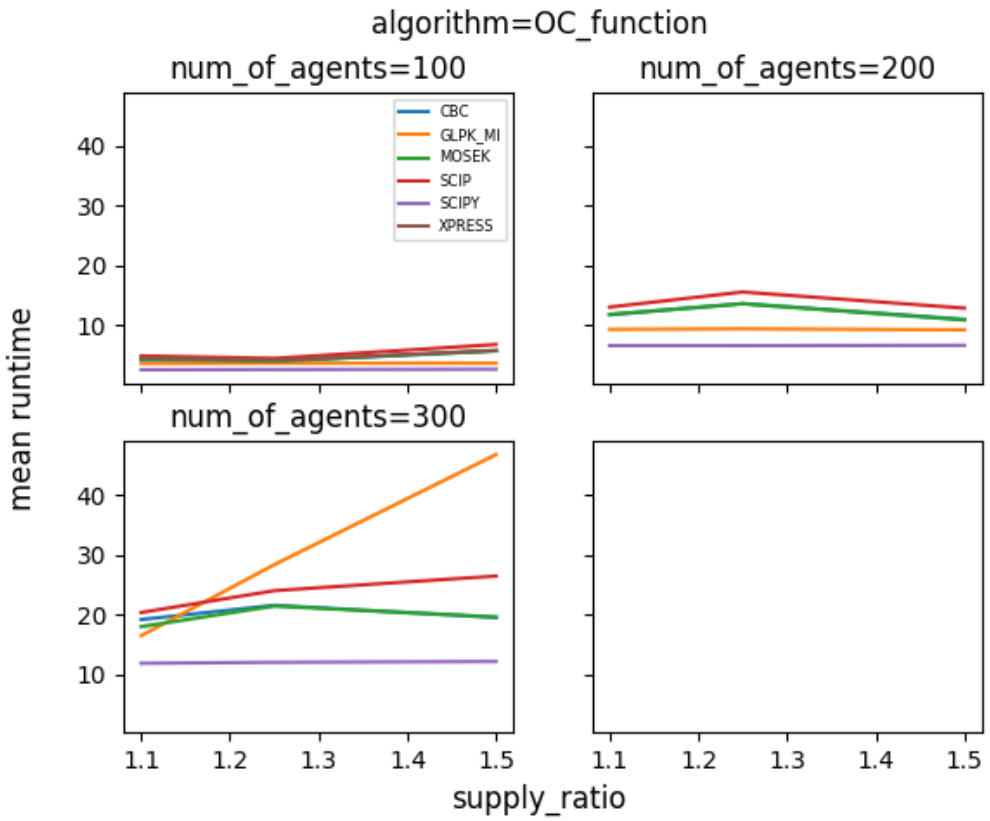
Interpretation:

1. SCIPY leads with the lowest runtimes across all scenarios, maintaining efficiency.

2. CBC shows a dramatic rise in runtime at 300 agents, making it less suitable for large-scale problems.

3. SCIP and MOSEK exhibit consistent runtimes, with slight increases as the supply ratio and agent count grow.

OC:



Interpretation:

1. SCIPY consistently has the lowest runtimes across all supply ratios and agent counts, demonstrating excellent efficiency.

2. GLPK_MI and MOSEK show moderate performance, with runtimes increasing slightly as the number of agents grows.

3. CBC experiences a significant runtime increase with 300 agents, indicating potential inefficiency with larger problems.

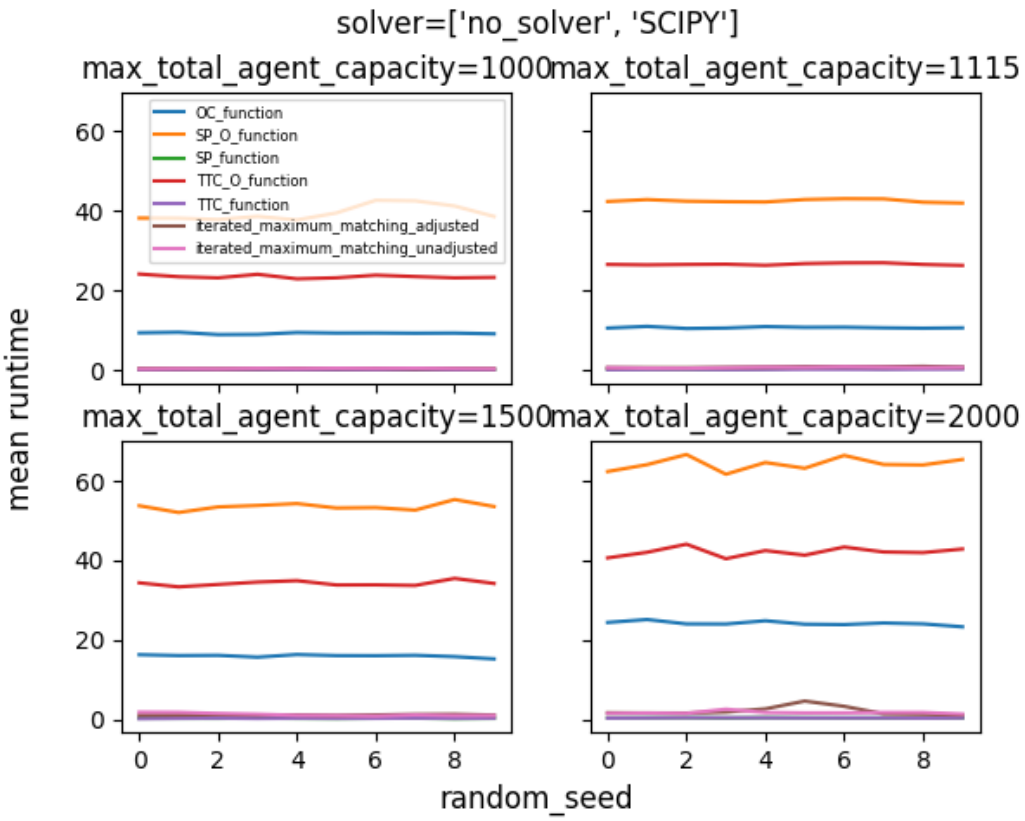
4. XPRESS is stable and efficient but not as fast as SCIPY.

Summary: SCIPY is the most efficient solver across all algorithms and scenarios, while CBC struggles with larger problems, particularly at higher supply ratios. Other solvers like MOSEK and SCIP offer consistent but slower performance compared to SCIPY.

ARIEL

1. **Runtime Plot**
   Overview: These plots show the mean runtime of all the algorithms as a

function of the max total agent capacity, for different random seeds.



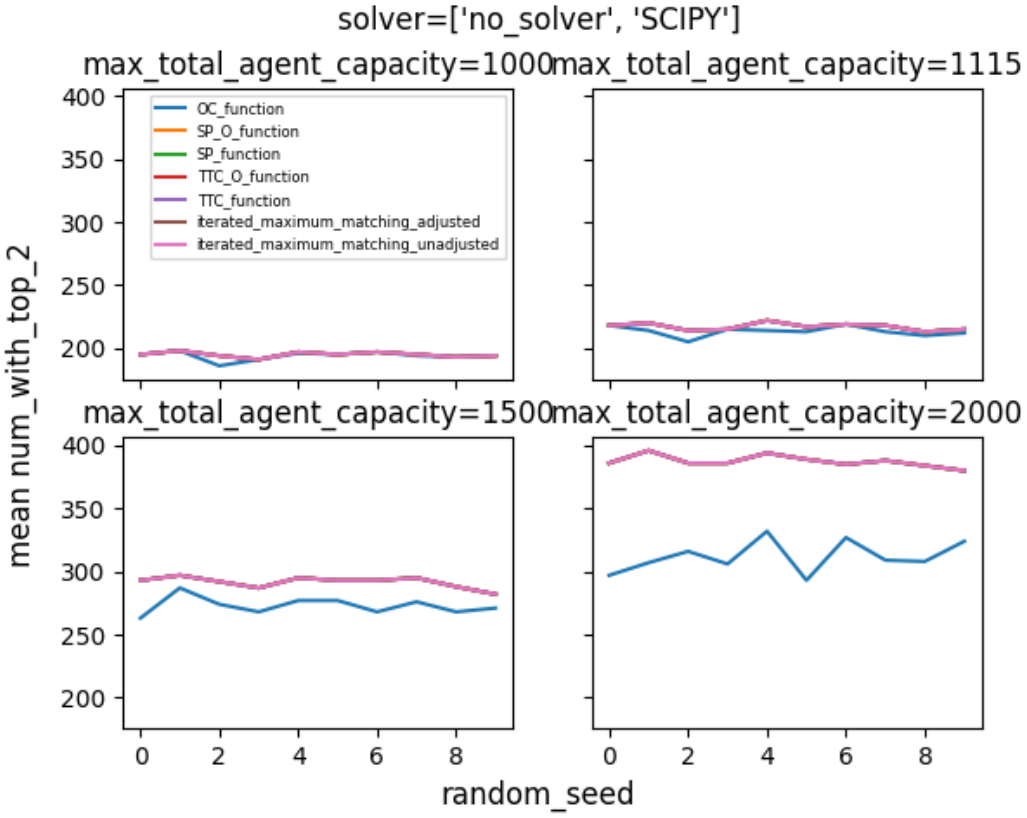
Interpretation: The runtime increases as the maximum total agent capacity increases, with OC_function, TTC_O_function and SP_0_function showing consistently higher runtimes.
The iterated matching algorithms (adjusted and unadjusted) remain efficient across different capacities and seeds.
The effect of random seed on runtime is minimal, indicating the stability of runtime performance across different scenarios.

2. **Top 1, Top 2, and Top 3 Satisfaction**
   Overview: These plots display the number of agents who received one of their top 1, top 2, or top 3 choices.

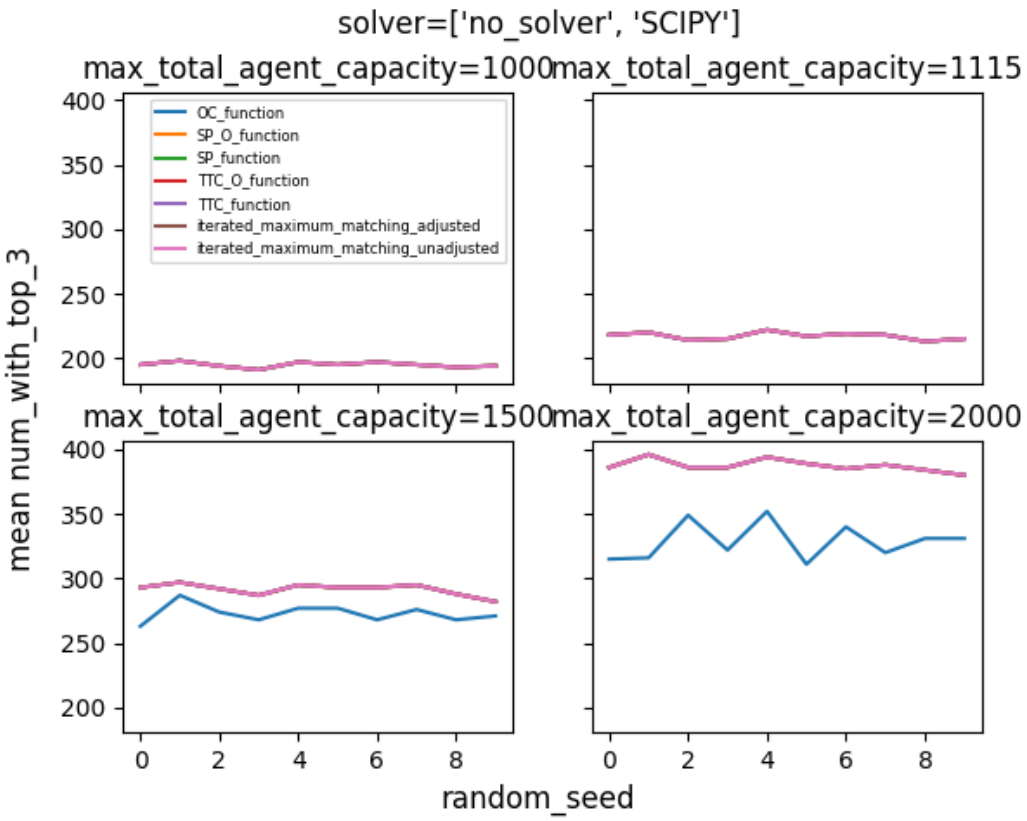
Top 1:


solver=['no_solver', 'SCIPY']
max_total_agent_capacity=1000
max_total_agent_capacity=1115
max_total_agent_capacity=1500
max_total_agent_capacity=2000
OC_function
SP_O_function
SP_function
TTC_O_function
TTC_function
iterated_maximum_matching_adjusted
iterated_maximum_matching_unadjusted
mean num_with_top_1
random_seed

Top 2:


solver=['no_solver', 'SCIPY']
max_total_agent_capacity=1000
max_total_agent_capacity=1115
max_total_agent_capacity=1500
max_total_agent_capacity=2000
OC_function
SP_O_function
SP_function
TTC_O_function
TTC_function
iterated_maximum_matching_adjusted
iterated_maximum_matching_unadjusted
mean num_with_top_2
random_seed

Top 3:

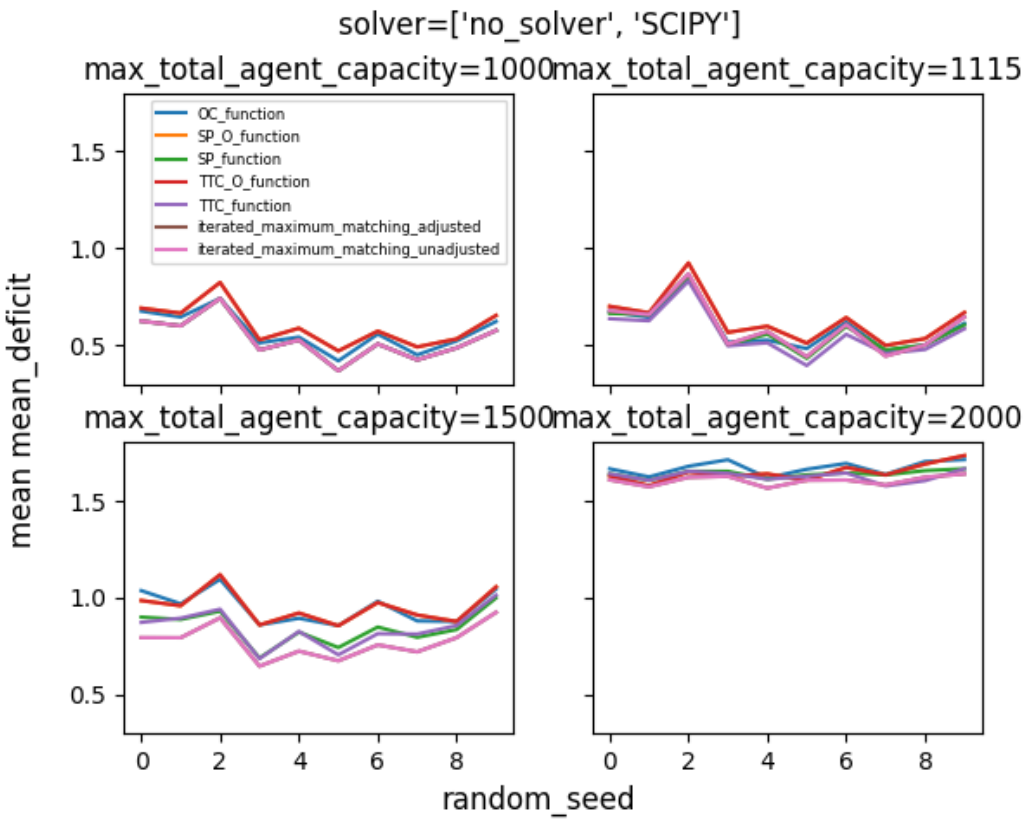


Interpretation: OC_function show more variation and lower satisfaction, especially in higher capacities then all the other algorithms

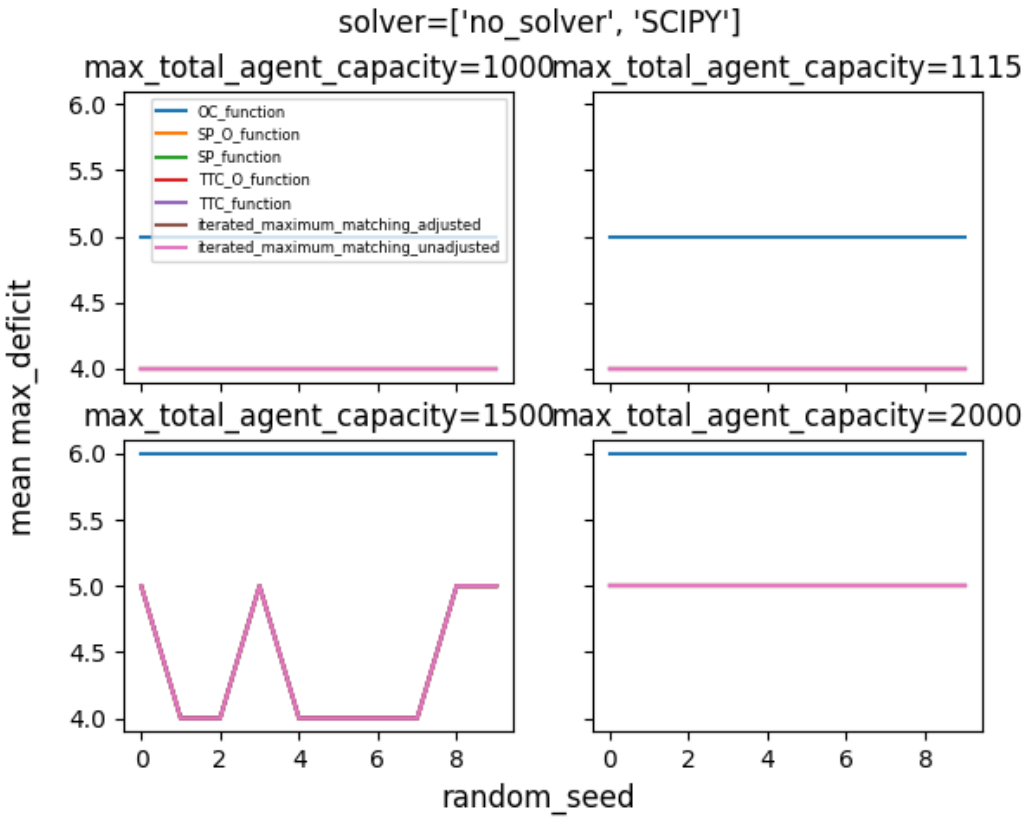
3. **Mean and Max Deficit**

Overview: These plots measure the mean and maximum deficit in the allocation process as a function of the random seed.

mean deficit:



max deficit:



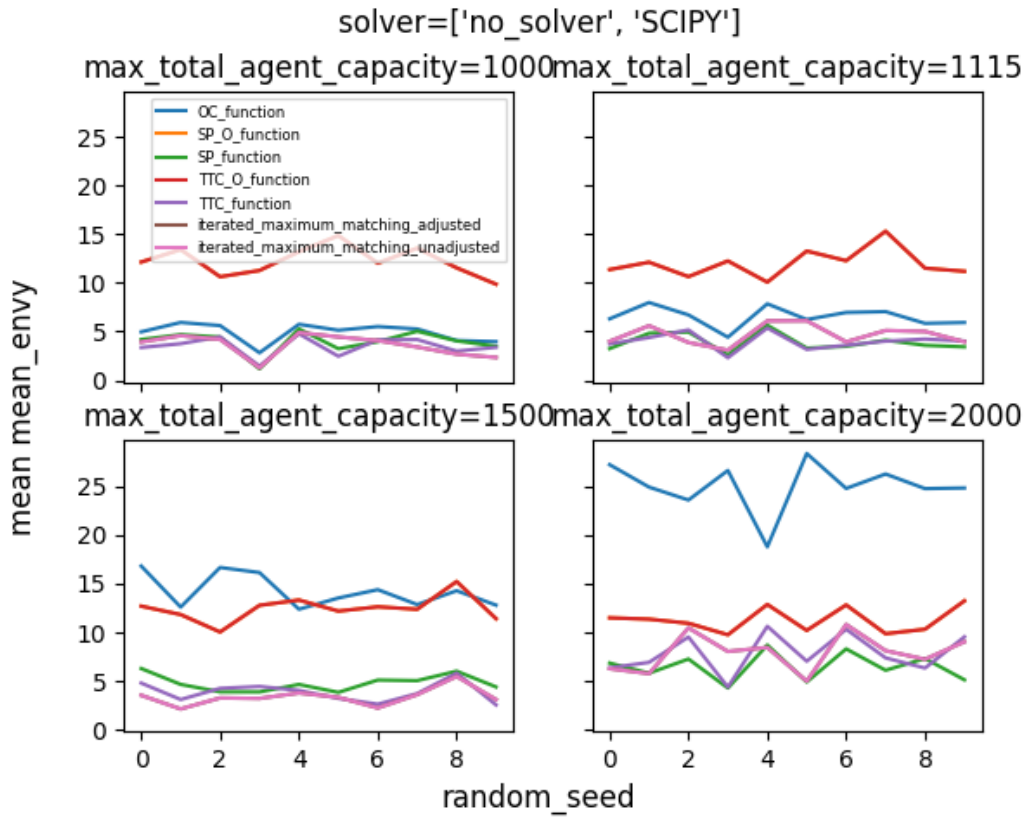
Interpretation: The iterated matching algorithms generally have low deficits,

though there are occasional spikes with certain random seeds.
OC_function shows a higher and consistent deficit across different capacities.
The random seed seems to have a more significant impact on mean deficit than on maximum deficit.

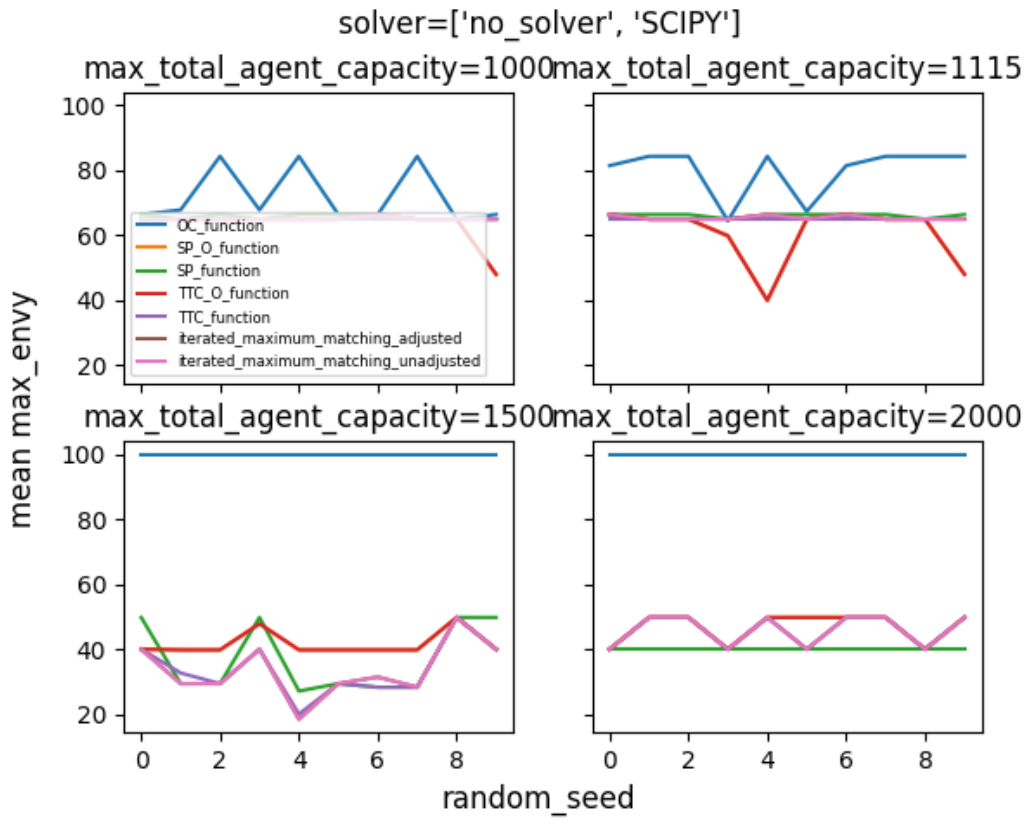
4. Mean and Max Envy

Overview: These plots represent the level of envy, with lower values indicating a more fair allocation.

mean envy:



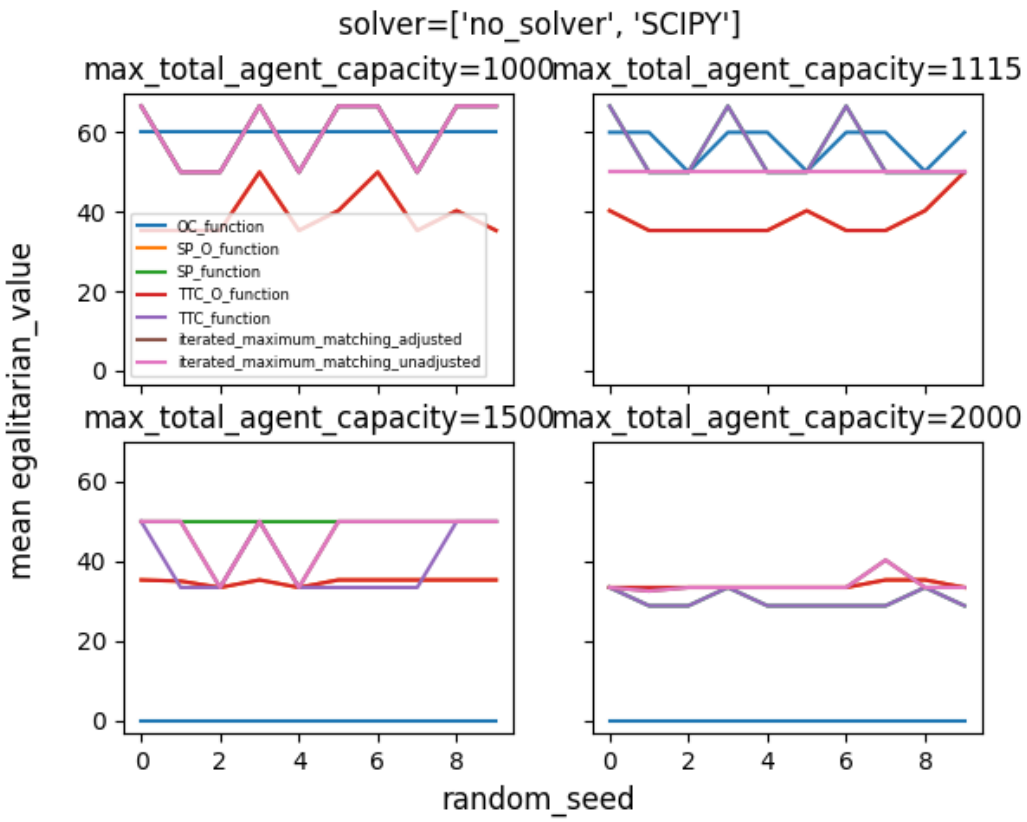
max envy:



Interpretation: OC_function show higher envy, especially in higher agent capacities, which suggests less fair allocations.
Iterated matching algorithms generally maintain lower envy levels, with some variation depending on the random seed.
Maximum envy tends to fluctuate more than mean envy, indicating occasional extreme cases where envy is high.

5. **Egalitarian Value**

Overview: This plot shows the mean egalitarian value, representing the fairness of the allocation.

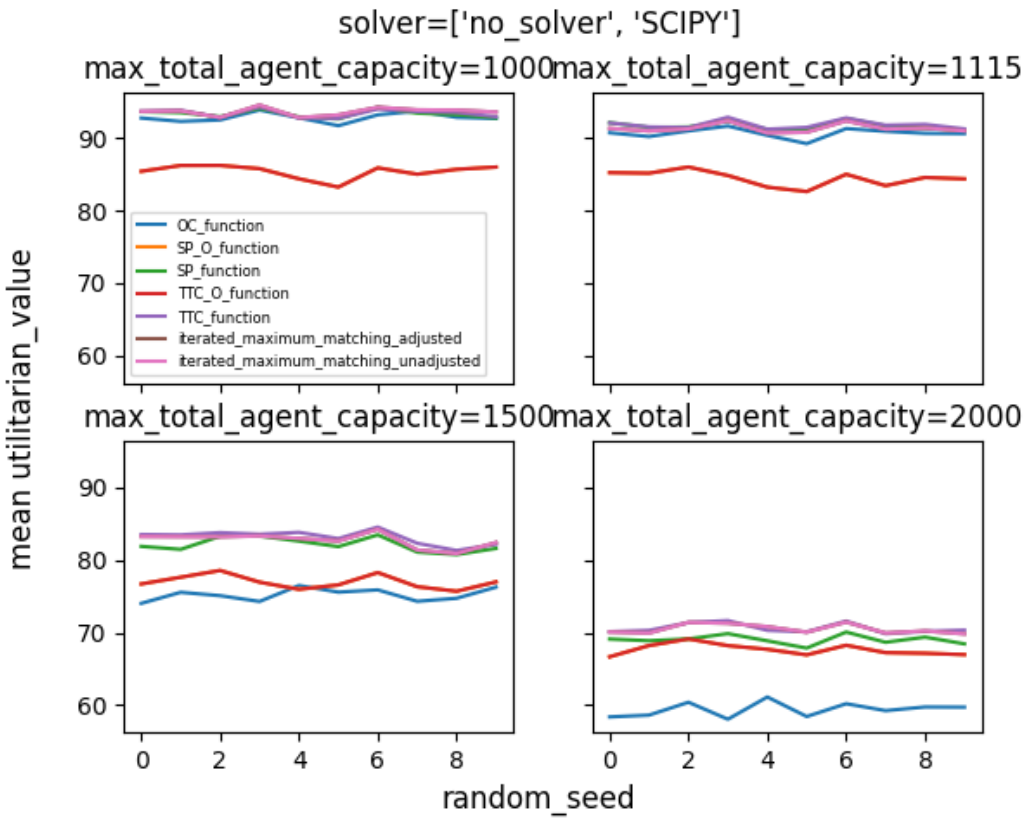


Interpretation: Iterated matching algorithms provide higher egalitarian values across different random seeds and agent capacities, reinforcing their effectiveness in fair allocation.
OC_function again lag behind, showing lower fairness values.
There is some fluctuation in egalitarian value based on the random seed, indicating that randomness plays a role in fairness outcomes.

6. **Utilitarian Value**

Overview: This plot displays the mean utilitarian value, representing overall

welfare or satisfaction.

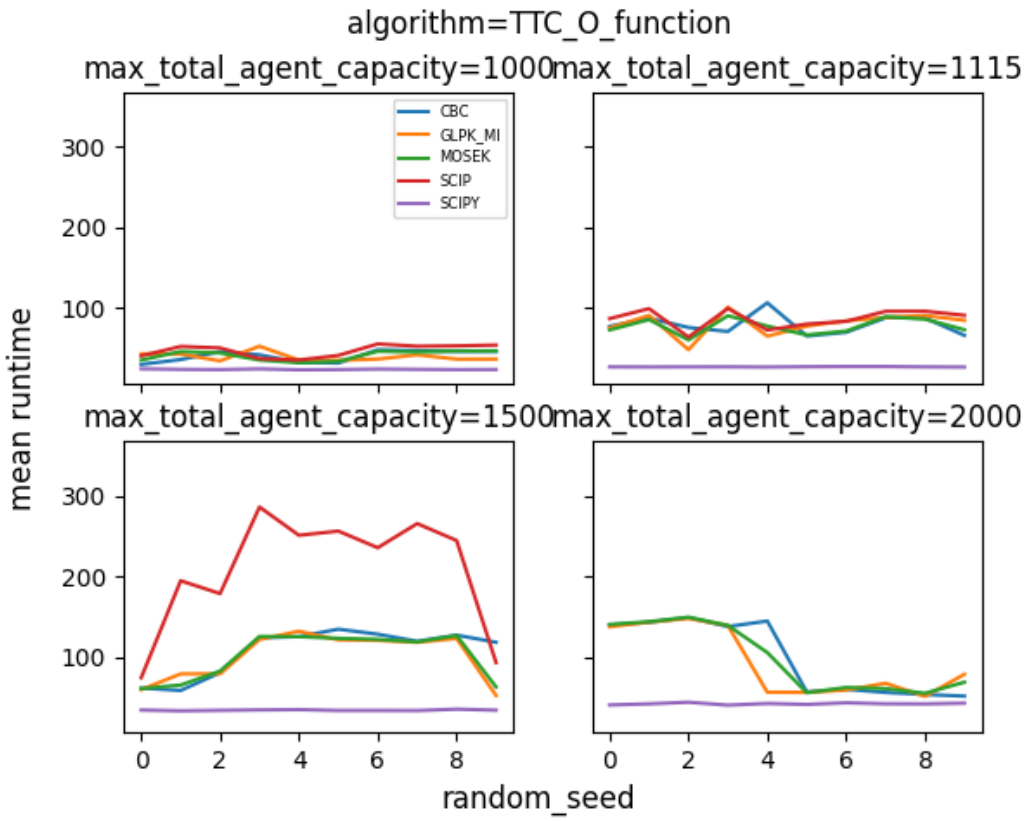


Interpretation: Utilitarian value remains relatively stable across different random seeds, with iterated matching algorithms and TTC_function performing better. influence of the solver appears to maintain high utilitarian values even as agent capacity increases.
OC_function tend to show lower utilitarian values, indicating less overall satisfaction.

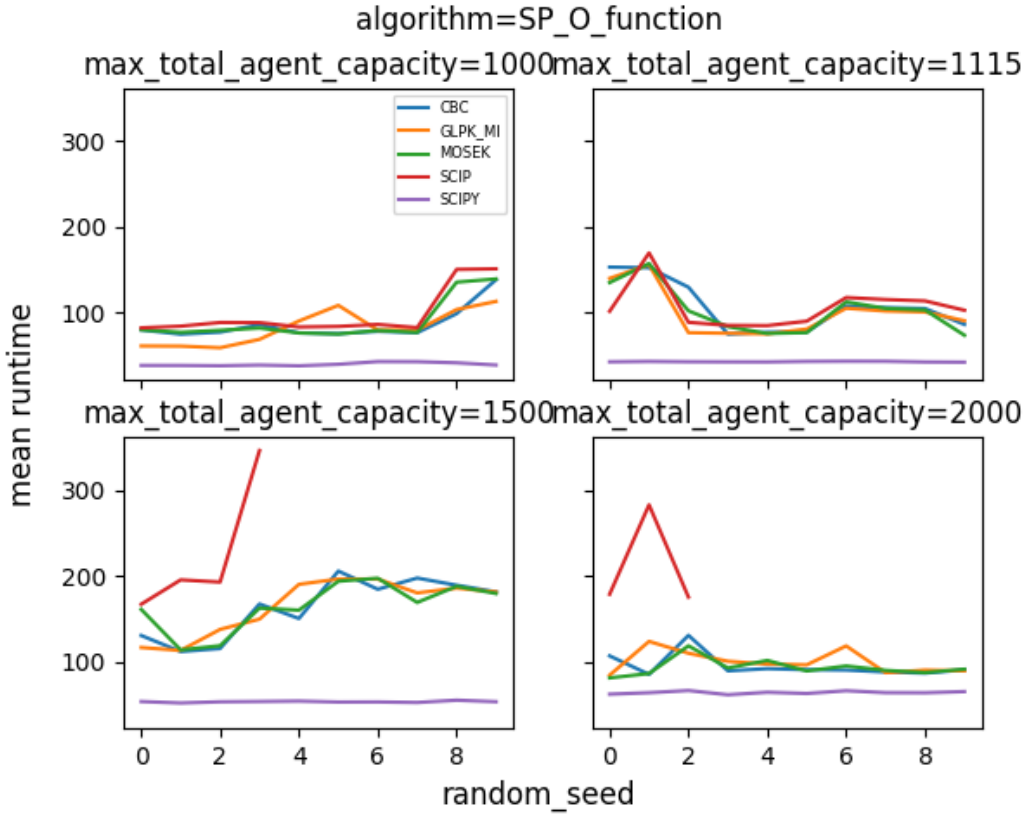
7. Solvers

Overview: To enhance the performance of our algorithms, we implemented the option for users to select a solver from the cvxpy library. This plot illustrates the average runtime of each solver when applied to the optimization algorithms.
solver we used**:** CBC, GLPK_MI, MOSEK, SCIP and SCIPY

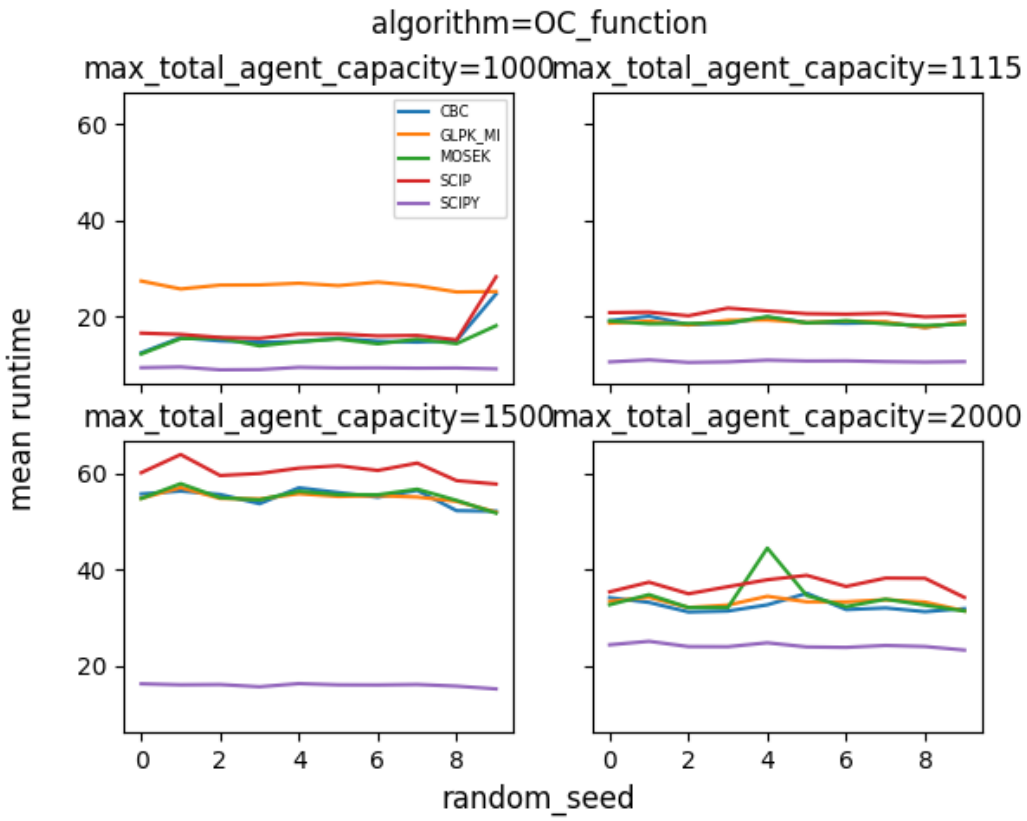
TTC-O:


algorithm=TTC_O_function
max_total_agent_capacity=1000
max_total_agent_capacity=1115
max_total_agent_capacity=1500
max_total_agent_capacity=2000
CBC
GLPK_MI
MOSEK
SCIP
SCIPY
mean runtime
random_seed
100
200
300
0
2
4
6
8

SP-O:


algorithm=SP_O_function
max_total_agent_capacity=1000
max_total_agent_capacity=1115
max_total_agent_capacity=1500
max_total_agent_capacity=2000
CBC
GLPK_MI
MOSEK
SCIP
SCIPY
mean runtime
random_seed
100
200
300
0
2
4
6
8

OC:



Interpretation:

1. SCIPY consistently has the lowest runtimes across all algorithms (OC_function, SP_O_function, and TTC_O_function), making it the most efficient solver.

2. SCIP shows significant variability, particularly with SP_O_function and TTC_O_function, sometimes leading to high runtimes, especially at larger capacities.

3. GLPK_MI, MOSEK, and CBC are stable but have higher runtimes than SCIPY, making them reliable but less efficient alternatives.

Summary: SCIPY is the best choice for efficiency across all tested scenarios, while SCIP may struggle with specific cases, and other solvers provide consistent but slower performance.

Research conclusions

conclusions

Initially, based on our guiding paper and prior to coding, we hypothesized that the OC algorithm would be the most efficient. Analyzing the results of the article, which examined various aspects, it was often pointed out that the OC algorithm outperformed the other five algorithms. These tests provided a strong basis for our initial assumption, as they demonstrated the superior performance of the OC algorithm in several scenarios.

However, after implementing the algorithms and running them on large datasets, our analysis revealed different results. When evaluating the running time, the OC algorithm did prove to be the most efficient. Additionally, in terms of bid utility maximization, the OC algorithm slightly outperformed the others. These initial results seemed to confirm the expected dominance of the OC algorithm.

Surprisingly, in other tests, the TTC-O algorithm significantly exceeded our expectations, outperforming the OC algorithm by a considerable margin. These tests included scenarios that assessed the fairness of the algorithms under varied conditions. The TTC-O algorithm has demonstrated exceptional versatility and robustness, making it a strong contender in situations where fairness and equity are critical considerations.

We also examined a dimension that was not addressed by the authors of the article: the concept of "envy" in algorithms. Envy, in this context, occurs when a student with more points than another expects the courses, they have chosen to be more beneficial. Our findings revealed that the OC algorithm performed worse in this regard compared to the other algorithms, highlighting a significant drawback not discussed in the paper.

Further analysis of the algorithmic performance in the mean and max deficit aspect reinforced this observation. We found that the OC algorithm performed worse than the other algorithms in these metrics. The significant gap in these metrics prompted us to reexamine our application, as such a large gap indicated a potential problem. After further investigation, we identified that this gap was mainly due to a subset of students who did not receive any assignments in the courses, which led to increased deficit compared to the other algorithms. This raised the question of whether this result was a logical consequence of the design of the OC algorithm or whether it indicated a flaw in our implementation.

To investigate further, we conducted tests with 300 students and 25 courses. The total number of available seats in all courses was 1,000, compared to the 1,800 seats demanded by the students (since each of the 300 students chose six courses). Out of all the students, seven were left without any course assignment. We then focused on comparing the results of student S5, who received no

courses, with those of student S112, who received six courses (achieving 89% of the maximum possible offers), and student S150, who also received six courses (achieved 74% of the maximum possible offers, the lowest percentage of those who received six courses).

Given that the OC algorithm favors ranking over bids, we tested both metrics:

Total offers for S112: 418. If S5 had been assigned the S112 courses, their total offers would have been 213.

S112 overall rating: 121. If S5 had taken these courses, their overall rating would have been 72.

Total offers for S150: 334. If S5 had been assigned the S150 courses, their total offers would have been 207.

S150 overall rating: 110. If S5 had taken these courses, their overall rating would have been 68.

These results, along with a detailed analysis of the algorithm's logs, revealed that the OC algorithm does prioritize maximizing the benefit derived from the bids, often at the expense of fairness. This finding highlights a significant flaw in the OC algorithm- its emphasis on bid maximization can lead to the neglect of fair treatment among students.

The noticeable gap in the deficit highlights the need to critically evaluate the design of the OC algorithm. This disparity raises questions about applicability in scenarios where fairness is as essential as efficiency. Our findings suggest that although the OC algorithm excels at maximizing the utility of bids, its potential to foster unfair outcomes makes it less appropriate in contexts where the desire for a fair distribution is a priority.

difficulties

One difficulty arose from the similarities and differences between the SP-O and TTC-O algorithms. Although these algorithms share some basic principles, we expected their results to be different. At the beginning of the paper, the authors of the paper provided small-scale examples to illustrate each algorithm. In these small samples the results for SP-O and TTC-O were similar. The authors noted that this similarity is due to the limited size of the samples. But even when the authors ran large-scale tests involving 900 students and ran each simulation 100 times under varying conditions, the results for SP-O and TTC-O were consistently similar.

Another significant challenge occurred during the initial thinking of how to implement the SP-O algorithm. Initially, when we did the manual tests, we approached the algorithm by calculating the optimal division for each round and allocating the price of the course according to the offers given by the first student who did not accept this course, and later deducting this price from the course.

Cumulative suggestions from the students who took this course, this thought arose from our expectations that the implementation of SP and SP-O would be similar. This approach led to unexpected and illogical results, where students have negative suggestions - something that should not happen within the rules of the algorithm.

Upon closer examination, we realized that our misunderstanding stems from the calculation of the course price.

This discrepancy highlighted a possible misunderstanding of the implementation of the algorithm.

In light of this, we modified our approach to the implementation of the SP-O algorithm by implementing the algorithm using the equations detailed in the paper. This adjustment led to a new set of challenges.

One of the linear programming equations included a variable p that should calculate the price for each course. However, upon implementing the algorithm, we found that the calculated price was consistently tiny—so small that the system treated it as zero. This result reduced the distinction between SP-O and TTC-O, and casts doubt on the apparent innovation of the SP-O algorithm.

After analyzing these findings, we gained insight into the identical results presented in the lectures detailed in the article. This understanding led us to contact the authors of the article for further clarification. Unfortunately, despite our efforts, we have not received a response from them until today.

Lessons learned

The project was a deep learning experience that extended beyond the technical challenges of implementing algorithms. We learned the importance of questioning assumptions, even those based on published literature. Our initial confidence in the superiority of the OC algorithm, based on the paper's conclusions, was challenged by our findings.

Moreover, the difficulties we encountered with the SP-O algorithm highlighted the complexity inherent in algorithmic implementation. The gap between theoretical descriptions and practical application became apparent, especially when our initial approach led to unintended and illogical results.

In addition, the challenges associated with communication with the authors of the original article highlighted the importance of collaboration and the potential

obstacles when seeking clarification from external sources. Although we have yet to receive answers to our inquiries, the process of analyzing and adjusting our approach in the absence of guidance has been an invaluable part of our learning journey.

Finally, our investigation into the concept of envy within the algorithms provided us with new insights into fairness. By investigating aspects not originally covered in the paper, we have contributed to a broader understanding of the effects of the algorithms, paving the way for further research and refinement.

In conclusion, this project was not just an exercise in applying algorithms but a comprehensive learning experience that challenged our assumptions, tested our problem-solving skills, and deepened our understanding of algorithmic fairness and efficiency. The lessons learned will undoubtedly inform our future work and contribute to our continued development as computer scientists.