OPTIMIZATION METHODS APPLIED TO BIPARTITE MATCHINGS: ASSIGNING STUDENTS TO SCHOOLS MAXMIZING FOR STUDENT SUCCESS

A PREPRINT

Sabrina Enriquez
Graduate Group in Biostatistics
University of California, Davis
Davis, CA 95616
seenriquez@ucdavis.edu

March 15, 2021

ABSTRACT

Bipartite matching problems are ubiquitous in every day life and are therefore a necessary class of problems to solve with decent efficiency. A bipartite matching problem is characterized by having 2 distinct sets of nodes that we wish to connect using a single edge between them. There are examples in nearly every industry but some frequently explored scenarios are matching applicants with jobs, patients with transplants, and students with schools. In this project we will follow methods outlined in "Migration as Submodular Optimization" Gölz and Procaccia, 2019 to match students to schools based on the student's and school's preference rankings. Using a "school choice" data set provided by Goto, Kojima, Kurata, Tamura, and Yokoo, 2019, we will employ the matching methods demonstrated in Gölz and Procaccia, 2019 to match students with schools by maximizing the likelihood that they will be offered admission into a school's Honors Program A or Honors Program B. We will consider a student "successful" if they are admitted into one of the programs at a school they are matched to. Our data has a student component where each student ranks their preference for a school's Honors Program A and each school ranks their preference for each student's admission to Honors Program B. This two- sided preference scenario is commonly referred to as the "Stable Marriage Problem" popularized by Gale and Shapley, 1962.

Keywords bipartite matching · submodular · optimization · matching theory

1 Motivation

In "Migration as Submodular Optimization" Gölz and Procaccia, 2019 build on existing algorithms to match migrants to localities while maximizing the probability of employment in a given locality. By maximizing for employment the classic stable marriage problem is given an extra layer of constraints and therefore leads to better matchings. They explain that they are motivated by assigning matches based on the likelihood of migrant success, and in this project we will match students to schools from a the data set provided by Goto et al., 2019 with similar motivations. Rather than maximizing for probability of employment as in the migrant-locality scenario, we will maximize the probability of academic success, defined by admission into special honor programs. Gölz and Procaccia, 2019 argue that previous methods for maximizing for employment likelihood relies on a strong additivity assumption which does not account for competition effects. Similarly, academic success of a given student is also held subject to competition effects when we rank students against each other. This consideration is very important when considering a "good" match for high school students since college admissions and scholarship opportunities are regularly determined by class ranking and whether or not they can take honors courses.

An example of the importance of choosing a high school based on predicted class ranking is the University of California "Statewide guarantee", 2020, which states "If you're in the top 9 percent of California high school graduates and aren't

admitted to any of the UC campuses you apply to, you'll be offered a spot at another campus if space is available". With the UC admission promise, as well as other similar admissions policies in other states, our problem of choosing a suitable match for as many students as possible is a heavy burden that should be given utmost importance. With these high stakes in mind, we venture forward in using the techniques described by Gölz and Procaccia, 2019 to assign students to schools while taking into account their preferences, the schools' preferences, and competition effects for students.

2 Data Overview

Goto et al., 2019 provided the data we are considering for this student- school pairing. We have two data sets to consider: student data and school data.

Student data has 875 row observations corresponding to each student and 50 columns corresponding to each school. The student data set contains the rankings each student gave to each of the 50 schools and some of those rankings have duplicates i.e. a student may have ranked two schools ranking=7 and no schools ranking=1. Thus, the data is not perfect, but we do not require distinct preferences for this matching algorithm.

School data has 50 row observations corresponding to each of the 50 schools and 875 columns representing each of the students. The school data set contains the rankings that every school gave to each student and once again has duplicate rankings.

Each of these data sets only have one set of rankings and since we are determining the matching by maximizing for two possible "successful" outcomes– getting into honors A or honors B– we use the average of the school-provided rankings of students as the ranking for honors A and the average of the student-provided rankings of schools as the ranking for honors B. We interpret the rankings as highest is most desirable and lowest is least desirable. Define the 50×875 matrix of school data as $school[l, i]$ where each value is the ranking school l gave to student i for honors program A. Define the 875×50 student data matrix as $student[i, l]$ where each entry is the ranking student i gave to school l for honors program B. We find the corresponding probabilities of admission by: given student i and school l, let $rA_(i) = \frac{1}{50}\sum_{l=0}^{49} school[l, i]$ be the average ranking of student i by schools l for honors A. Let $rB_(i) = \frac{1}{50}\sum_{l=0}^{49} student[i, l]$ be the average ranking for program B by student i over all schools. Then since all rankings are between 0 and 874 for school data and 0 and 49 for student data, their averages are in that range and the probabilities of admission into each program at some school are:

$$P_{(i)}(A) = P_{(i)}(\text{Admission into Honors A (anywhere)}) = \frac{1}{875} rA_{(i)} \tag{1}$$

$$P_{(i)}(B) = P_{(i)}(\text{Admission into Honors B (anywhere)}) = \frac{1}{50} rB_{(i)}. \tag{2}$$

Meanwhile, the probability that student i is admitted into honors A at school l is:

$$P_{(i,l)}(\text{Admission into Honors A}) = \frac{school[l, i]}{875}.$$

Lastly, the probability that student i is admitted into honors B at school l is:

$$P_{(i,l)}(\text{Admission into Honors B}) = \frac{student[i, l]}{50}.$$

The algorithms we are implementing require that all of the schools have equal capacities and since 50 does not evenly divide 875 we are forced to modify our data to run the proposed algorithm. We decided to truncate the data at 850 students creating a constant capacity of 17 for each of the 50 schools. By doing this we have sacrificed including 25 students in this matching process and have slightly changed the competition aspect since the students are ranked 0 through 874 by each of the 50 schools. When we truncate the data we do not make any intentionally biased alteration to the rankings, but it does slightly change the average ranking for each student: $rA(i)$ and $rB(i)$. This difference is statistically insignificant given the size of our data so we deem this truncation acceptable. Furthermore, to include the 25 students we leave out we could run the algorithm again with the front half of our data truncated and take the intersection of our final matchings to determine a matching that is optimal given our algorithm and includes all students. We would need to impose decision rules for taking the intersection and consider certain edge cases, but in this project we keep our scope focused on solely implementing the algorithm on the 850 students with the certainty that we are not inappropriately altering the results.

3 Methodology

We employ the following methodology described by Gölz and Procaccia, 2019 and tweak it to suit our problem:

The **objective function** f receives a set of student-school pairs as input, and returns the predicted number of "successful" students under the corresponding assignment. We define "success" for a student as being admitted to honors program A or honors program B at the school they are matched to. We assume that f is (monotone) submodular: individual elements provide diminishing marginal returns. If S and T are two subsets of student- school pairs such that $S \subseteq T$ and (i, l) is a student-school pair such that $(i, l) \notin T$, then

$$f(S \cup \{(i, l)\}) - f(S) \geq f(T \cup \{(i, l)\}) - f(T).$$

By assuming submodularity, we impose that for the larger number of students competing with i for an honors program admission within a school l, the less likely it is that i will be successful in achieving admission. Moreover, this means smaller marginal contribution of (i, l) to the objective function– the overall number of "successful" students. We relate our problem to the original migrant-locality setup by defining the following correspondences:

- number of agents = number of students $N = 850$
- number of localities = number of schools $L = 50$
- locality capacity (cap) = school capacity: 17
- competence = $P_{(i)}(A) + P_{(i)}(B) - P_{(i)}(A) * P_{(i)}(B)$ probability of admission to A or B.
- profession 1 $\rightleftarrows$ talents for honors program A
- profession 2 $\rightleftarrows$ talents for honors program B
- jobs in profession 1 $\rightleftarrows$ spots in honors program A
- jobs in profession 2 $\rightleftarrows$ spots in honors program B

In the original paper Gölz and Procaccia, 2019 distinguish between "skilled jobs" and "unskilled jobs" and make skilled jobs more competitive, but we do not impose a ranking on the honors programs. The algorithm we are modifying for our data is provided by the author at Gölz, 2019. The source code provided is written to run simulations on randomly generated data rather than provided data so we alter the code to work on our data and provide our altered version in the appendix.

As explained in the paper, the objective function is relaxed to be *approximately* submodular since the "success" estimation introduces noise which can violate the conditions of strict submodularity. We choose to maximize for two honors programs to follow the framework of the original paper, because with matching migrants to locality with two possible professions as a constraint, Gölz and Procaccia, 2019 showed that "accounting for submodularity almost always outweigh the loss associated with using an inexact optimization technique". They demonstrated their method's improvement by comparing their submodular model with the baseline approach of assuming additivity across three settings: a retroactive- correction model, an interview model, and a coordination model. They were able to prove that the worst case guarantees given by the greedy algorithm hold when accounting for submodularity and they found impressive performance bounds for the greedy algorithm given these submodular constraints Gölz and Procaccia, 2019(Theorem 3.1).

4 Basic theory used

This variation of the classic Stable Marriage problem can be thought of as the intersection of two matroids which unlocks results and tools that make their optimization provably efficient. First we define a matroid:

Definition 4.1 (Matroid)**.** (Harary, 1994) A matroid consists of a finite set M of elements together with a family $C = \{C_1, C_2, ...\}$ of nonempty subsets of M, called circuits, which satisfy the axioms:

- No proper subset of a circuit is a circuit
- If $x \in C_1 \cap C_2$ and $C_1 \neq C_2$, then $C_1 \cup C_2 - \{x\}$ contains a circuit.

We can interpret matching each student to at most one school with the constraint induced by schools capacities as partition matroids over N students and L schools: $N \times L$. The following explanation is provided in the paper and reworded for our scenario: In the first matroid we restrict to a matching to select at most one edge out of the set of edges incident to each student i and in the second matroid, at most q_l edges out of the set of edges incident to each

school l. The matchings are exactly the sets that are independent in both matroids. Therefore, maximizing "success" (admission into an honors program) among matchings is an instance of maximizing a submodular function subject to multiple matroid constraints.

Definition 4.2 (Greedy Algorithm)**.** (Gölz and Procaccia, 2019) Call the matroids $\mathcal{M}_p = (N, \mathcal{F}_p)$ for $1 \leq p \leq P$, let sp^p denote the span with respect to $\mathcal{M}_p$, and let the intersection of the matroids be $\mathcal{F} := \cap_{p=1}^{P} \mathcal{F}_p$. To refer to the marginal contribution of an element j with respect to a set S, we set $\rho_j(S) := z(S \cup \{j\}) - z(S)$.

The greedy algorithm initializes $S^0 \longleftarrow \emptyset$, $N^0 \longleftarrow N$ and $t \longleftarrow 1$. Then it proceeds through the following steps, wherein t denotes the number of the current iteration:

Step 1: If $N^{t-1} = \emptyset$, stop with S^{t-1} as the greedy solution.

Step 2: Select $i(t) \in N^{t-1}$ for which $z(S^{t-1} \cup \{i(t)\})$ is maximal, with ties settled arbitrarily.

Step 3: If $S^{t-1} \cup \{i(t)\} \notin \mathcal{F}$, set $N^{t-1} \longleftarrow N^{t-1} \setminus \{i(t)\}$ and return to Step 1.

Step 4: If $S^{t-1} \cup \{i(t)\} \in \mathcal{F}$, set $\rho_{t-1} \longleftarrow \rho_{i(t)}(S^{t-1})$, $S^t \longleftarrow S^{t-1} \cup \{i(t)\}$, and $N^t \longleftarrow N^{t-1} \setminus \{i(t)\}$.

Step 5: Set $t \longleftarrow t + 1$ and continue from step 1.

With our problem cast as the intersection of two matroids, Gölz and Procaccia, 2019 proved that their three models 5 induce approximately submodular objective functions the following algorithmic bounds:

Theorem 4.1. Let $z : 2^N \longrightarrow \mathbf{R}$ be ϵ- approximately submodular and let $\hat{z}$ be the underlying monotone normalized submodular function, i.e. let

$$(1 - \epsilon)\hat{z}(S) \leq z(S) \leq (1 + \epsilon)\hat{z}(S)$$

for all $S \subseteq N$. Let $\mathcal{F}$ be the intersection of P matroids, and let k denote the size of the largest $S \in \mathcal{F}$. Then the greedy algorithm selects a set $S \in \mathcal{F}$ such that

$$(P + 1 + \frac{4\epsilon}{1 - \epsilon}k)\hat{z}(S) \geq \max_{S' \in \mathcal{F}} \hat{z}(S').$$

Using this background, we adopt their proofs for our scenario since they are exactly alike, and proceed with implementing their models and algorithms.

5 Models

In order to maximize for admission into an honors program over all student- school matchings we need a way of predicting the likelihood of admission under a given matching. The following models are three ways to predict the likelihood for success and are only based on parameters based on data. Their models do not make assumptions about the data and therefore ensure that the predictors "behave reasonably on all inputs and requires less data for fitting" Gölz and Procaccia (2019).

5.1 The Retroactive-Correction Model

(Gölz and Procaccia, 2019) This model is the easiest and generalizes the additive model used by Bansak et al., 2018 by retroactively correcting employment success for competition. In this setting students qualify for admission based on the average ranking for each student over all 50 schools and that number is corrected by a concave function. Assume N students can be partitioned into disjoint talents π, and that only students of the same talents compete for admission to honors programs. Each student i has the probability p_{il} of admission at school l. Referring to 2 this value is given by "competence".

The admission function at a school l and with a talent π (corresponding to honors program A or B at each school) is obtained by applying a concave and monotone *correction function* $\mathcal{C}_{l\pi} : \mathbf{N} \mapsto \mathbf{R}_{\geq 0}$ to the number of qualifying students. Total success is computed by the sum of success over all schools and talents/ honors programs. We use $x \mapsto \min(x, c)$ as our correction function for some constant c for each honors program A and B at every school. The constant c is the number of honors spots at a school in either honors program A or B. It represents the hard capacity for qualified students in each program at each school. The submodular function to optimize for is the expected success generated by this process. There is no direct competition between students of different talents but for each school there is a set capacity of 17 so they are not independent. Moreover, since the schools already ranked all of the students there is implicit competition in their "competence" probability. Here our model differs from the original paper because

rather than assigning random probabilities to qualification probabilities, we used the data for both programs (considered independently) to find the qualification probability.

In this setting the expected admission function induced by the model is submodular. Further explanation can be found in the paper by Gölz and Procaccia, 2019 Section 4.1.

5.2 The Interview Model

This model predicts success for honors admission given students apply for the programs in a random sequential order. We kept this sequence random as in the original paper as it did not make sense to use the data for this scenario. This model differs from the corrective model because success is not determined by a single hurdle of qualification but through a sequence of applications to individual honors programs. Each school has a certain number $k_{l\pi}$ spots in their honors programs for talent π, and each student $i \in \mathbf{N}$ has the probability p_{il} of admission for a particular honors program (A or B) at school l.

Admission to an honors program is calculated individually for each school l and talent π. Initially there are k_{il} spots available and then we iterate over students of talent π matched with school l in an order selected uniformly at random. For student i we flip p_{il} biased coins until we either hit success or fail $k_{l\pi}$ times. Each coin toss represents a student applying for a remaining spot in an honors program and if one succeeds then the student is considered "successful" and we decrement $k_{l\pi}$ and continue with the process for the next student. If in the end a student is not successful $k_{l\pi}$ remains unchanged for the next student.

In the interview model the *utility* of a matching is the sum over schools and honors programs of the expected number of successful students in that school and honors program.

Once again, the authors Gölz and Procaccia, 2019 proved that the expected admission function induced by the interview model is submodular (proof in their appendix) and therefore their algorithmic bound 4.1 holds.

5.3 The Coordination Model

Lastly, we consider the coordination model which determines compatibility between students and honors programs and then matches optimally. In Gölz and Procaccia, 2019 they make compatibility random but we once again calculate compatibility using the competence score given in 2. This model goes beyond strict separation of honors programs A versus B and assumes that success for admission for all students in the same school is coordinated.

In this setting each school l has a number of k_l honors spots and each student i has a certain probability p_{il} of being compatible for a specific honors program j. These probabilities might be induced by a strict partition into honors programs A and B, but can be less restrictive allowing students to be placed in to programs that are closer or further than their talents. This model is different from the interview model since in that model a student accepts admission to the first "successful" application, but now we determine compatibility between students and honors programs at school l and coordinate the assignment such that "success" for admission is optimized. This model makes sense if both honors programs are similar at every school and a student's talents are applicable to both programs A and B, but in the case that the talents and corresponding programs are extremely different, it would not make sense to use this model. For example, if the honors programs were in scuba diving and gardening a talented diver might not consider admission into a gardening program as a "success" and there is no reason to believe they would excel in that program versus being in no program at all.

To calculate "success" for admission at a school l we flap a coin with bias p_{ij} for each student i matched to l and each honors program j at l. By drawing an edge between each pair (i, j) whose coin flip succeeded, we obtain a bipartite graph. Admission success at this school is the size of a maximum matching, which corresponds to the highest number of students that can be given a spot in either honors program A or B.

The total *utility* of this matching is the the sum of expected admission over localities. For an interested reader, Gölz and Procaccia, 2019 provide a proof to their Theorem 4.3: "The expected employment [admission] function induced by the coordination model is submodular".

6 Computation and examples

In the original paper the authors ran experiments comparing their three models while varying 1: number of localities, 2: number of agents, 3: number of professions, and 4: job availability. Since we are implementing this framework for real data it does not make sense for us to vary the number of schools nor the number of students nor the number of

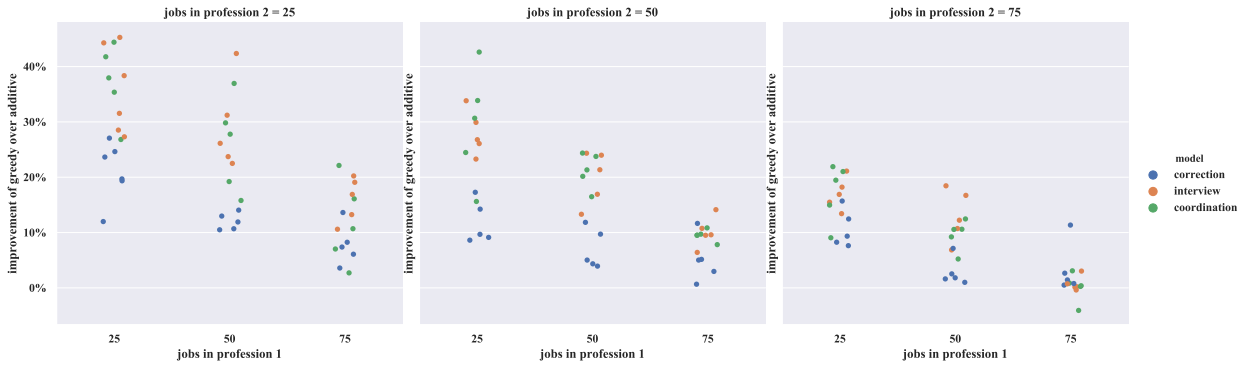


Figure 1: Gölz and Procaccia, 2019 results showing improvement of greedy over the additive approach in 3 settings with 3 different job availibilities for 2 professions.

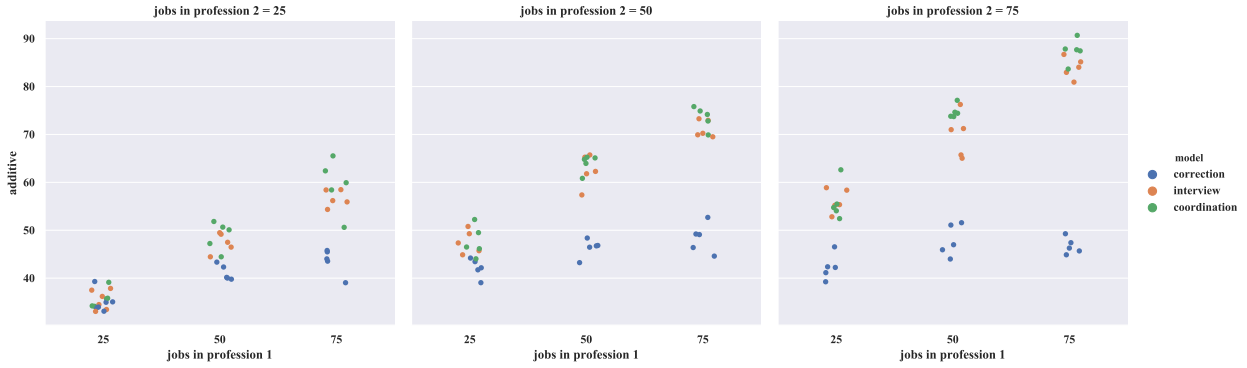


Figure 2: Gölz and Procaccia, 2019 results showing absolute utilities over the same settings and job availability varying for both professions

honors programs, but we will follow their analysis varying the total number of spots in honors A and honors B which corresponds to varying the number of jobs available.

Their results are summarized in figures 1 and 2. In their results (which we re-simulated using Gölz, 2019) we see that when there are fewer jobs the greedy approach outperforms the additive, but as the number of jobs available increases additive does well, as expected.

We use the same framework as the authors of this paper, but when running the computations it became clear that our data is too big and computing power too small to run the algorithm to match 850 students. In the appendix there is the code to execute it, but simply running the algorithm for one setting with 400 spots for each honors program took 4 hours and gave us a single data point shown in figure 3. So instead, we choose to truncate our data again at 100 students and match them to 10 schools with capacities 10. This computation still took about 4 hours to finish, but we were able to compute 5 data points for each model varying the honors A and B capacities from 25 to 50 to 75. We chose these numbers to mirror the original simulations and compare our findings.

In figure 5 we see the overall utility of the algorithm's matchings and find that our results mirror those of the original paper Gölz and Procaccia, 2019, affirming that accounting for competition effects increases utitily.

Moreover, in figure 4 we see that the greedy algorithm improves over the additive technique just as in the original paper's simulations, but there is a bit more spread especially for where there are only 25 spots in the honors programs as compared to the simulated data. Overall, applying the algorithm to our student- school matching data was successful in producing optimal "successful" matchings and reflected the theorems proven in Gölz and Procaccia, 2019.

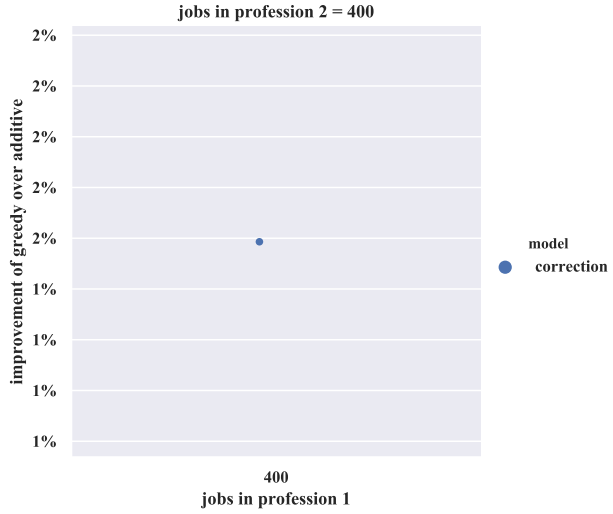


Figure 3: Improvement of greedy over additive in the correction model for 850 students, 50 schools and 400 spots in each honors program A and B (corresponding to "jobs in profession 1" and 2. This single computation took more than 4 hours.

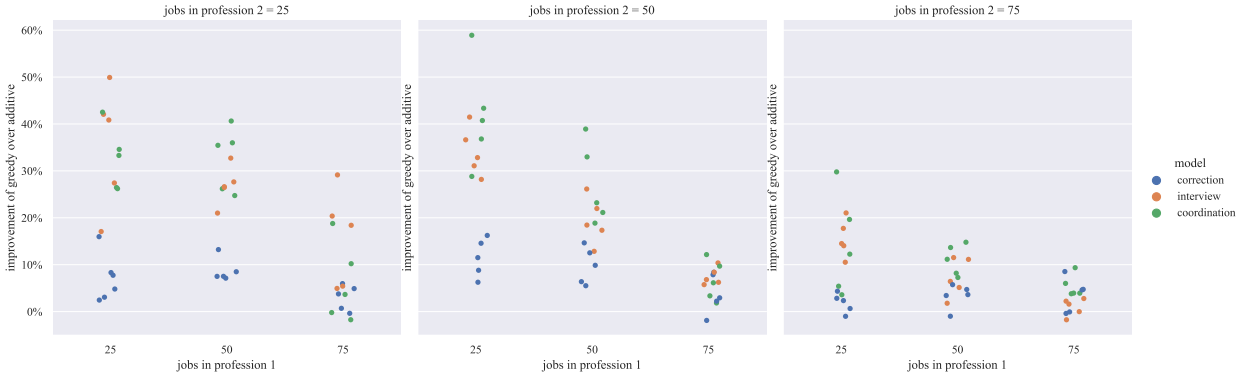


Figure 4: Improvement of greedy over additive for each setting with the number of avaiulable honors spots varying between 25, 50, and 75.

Figure 5: Absolute utility of matching students to schools determined by maximizing for admission into honors program A (corresponds to profession 1) and honors program B (corresponds to profession 2).

7 Conclusion

Using the algorithm provided by Gölz and Procaccia, 2019 we are able to account for competition effects and match 100 students to 10 schools maximizing for admission into an honors program. This optimization technique shows gains over the additive technique, but is computationally extremely expensive. We see that as the number of spots in honors programs increases the additive technique performs better and therefore we should use it when dealing with bigger data. However, we are able to affirm the findings from the paper and determine our matches better by using this new technique. Moreover, the worste case bounds for the greedy algorithm hold with submodular functions so if we have access to more computing power it may be worth it to run this on bigger data.

In the motivation section we stressed the importance of maximizing for student success when solving this bipartite matching problem and that importance generalizes to countless settings. For example, if we have a reliable model for predicting "success" of say an organ transplant or scarce medicine, this optimization technique is worth the computational expense to ensure optimal matching. Overall, we see that the techniques developed in Gölz and Procaccia, 2019 are applicable to many bipartite matching problems where there is a "success" factor and theoretically can be useful for a more nuanced situations where there are more than two honors programs or professions. However, it should be noted that such an exercise would require massive computational power and may take a very long time to run.

References

Bansak, K., Ferwerda, J., Hainmueller, J., Dillon, A., Hangartner, D., Lawrence, D., & Weinstein, J. (2018). Improving refugee integration through data-driven algorithmic assignment. *Science (New York, N.Y.) 359*(6373), 325.

Gale, D., & Shapley, L. (1962). College admissions and the stability of marriage. *American Mathematical Monthly*, *69*, 9–15.

Gölz, P. (2019). Migration repository. Retrieved from https://github.com/pgoelz/migration

Gölz, P., & Procaccia, A. D. (2019). Migration as submodular optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 549–556. doi:10.1609/aaai.v33i01.3301549

Goto, M., Kojima, F., Kurata, R., Tamura, A., & Yokoo, M. (2019). Replication data for: Designing matching mechanisms under general distributional constraints. Retrieved from https://www.openicpsr.org/openicpsr/project/114354/version/V1/view?path=/openicpsr/114354/fcr:versions/V1/Program/B.-Controlled-School-Choice-with-Hard-Minimum-and-Maximum-Quotas/Main&type=folder

Harary, F. (1994). *Matroids. graph theory.* Reading, MA: Addison-Wesley.

Statewide guarantee. (2020). Retrieved from https://admission.universityofcalifornia.edu/admission-requirements/freshman-requirements/california-residents/statewide-guarantee/

8 Appendix

In [1]:
```
from math import sqrt
from random import random, randrange, seed
```

In [2]:
```
import matplotlib
import seaborn
import pandas as pd
```

In [4]:
```
from models import *
from methods import *
```

In [5]:
```
seaborn.set(style="darkgrid")
matplotlib.rcParams["figure.dpi"] = 300
matplotlib.rcParams["font.family"] = "serif"
matplotlib.rcParams["font.serif"] = ["Times New Roman"]
```

In [ ]:
```

```

In [6]:
```
#import school and student choice data
schooldata = pd.read_csv('/Users/sabrina/Desktop/Winter 2020/258 Final P
roject/codeforfinal/schooldata.csv', header=None)
studentdata= pd.read_csv('/Users/sabrina/Desktop/Winter 2020/258 Final P
roject/codeforfinal/studentdata.csv', header=None)
```

In [7]:
```
schooldata
#schools 0 thru 49 = 50 schools
#rankings on students 1 thru 875
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	866	867	868	869	870	871	8
0	school 0:	314	437	195	270	100	388	768	95	193	...	540	302	760	233	695	851	6
1	school 1:	534	334	767	801	196	738	812	63	434	...	871	107	16	248	660	828	2
2	school 2:	657	348	674	622	708	331	283	55	466	...	137	614	550	541	854	335	2
3	school 3:	567	607	454	583	320	60	627	177	201	...	445	564	538	477	190	313	7
4	school 4:	845	814	265	477	84	855	85	408	298	...	102	115	50	234	691	416	2
5	school 5:	851	478	256	220	66	461	395	346	402	...	291	514	393	136	289	732	1
6	school 6:	70	429	257	863	597	340	843	421	364	...	438	462	238	784	392	192	3
7	school 7:	586	402	248	204	186	395	738	820	698	...	868	163	499	251	444	117	6
8	school 8:	53	584	807	36	294	662	602	649	235	...	550	295	765	700	110	337	2
9	school 9:	435	497	701	766	788	118	620	572	200	...	211	605	851	166	605	534	1
10	school 10:	17	595	125	324	837	660	128	7	513	...	510	663	563	548	723	598	2
11	school 11:	841	843	419	749	435	616	621	46	178	...	381	152	688	321	583	615	3
12	school 12:	283	84	537	557	742	592	453	862	46	...	451	738	583	518	669	553	7
13	school 13:	805	95	670	431	706	695	571	27	797	...	493	326	756	319	241	152	5
14	school 14:	311	370	533	866	225	192	452	381	319	...	353	239	776	753	319	291	5
15	school 15:	372	304	332	73	299	629	801	486	201	...	514	429	662	480	814	484	4
16	school 16:	341	147	778	702	755	138	39	300	377	...	329	393	232	598	457	259	2
17	school 17:	676	678	491	292	99	813	615	671	838	...	160	584	2	189	575	654	4
18	school 18:	281	601	466	203	336	624	290	298	394	...	473	154	355	38	538	869	2
19	school 19:	631	340	265	815	210	202	146	130	404	...	848	115	607	27	671	301	1
20	school 20:	379	609	667	535	756	392	192	425	338	...	335	485	209	542	86	425	3
21	school 21:	16	321	276	114	873	751	275	20	317	...	231	612	446	334	613	671	2
22	school 22:	874	55	102	566	500	667	713	382	291	...	188	385	703	218	464	714	6

	0	1	2	3	4	5	6	7	8	9	...	866	867	868	869	870	871	8
23	school 23:	571	310	584	190	765	678	534	401	76	...	154	431	764	847	232	723	4
24	school 24:	542	870	220	223	296	426	505	792	238	...	670	823	758	269	321	829	
25	school 25:	811	767	550	77	135	41	514	54	667	...	603	792	226	67	705	763	6
26	school 26:	32	724	32	847	662	171	410	773	421	...	326	148	803	455	565	52	1
27	school 27:	335	157	376	509	848	788	268	7	310	...	345	264	689	551	743	14	5
28	school 28:	448	353	646	63	245	666	93	281	70	...	845	478	17	136	133	274	2
29	school 29:	750	226	238	298	173	699	378	786	404	...	300	604	258	513	834	372	5
30	school 30:	199	164	176	716	95	724	859	738	22	...	682	296	570	673	622	337	2
31	school 31:	56	513	548	789	137	357	54	624	372	...	510	151	624	206	533	182	4
32	school 32:	873	467	523	482	769	600	31	718	342	...	816	774	124	394	633	285	3
33	school 33:	64	856	624	864	653	708	285	6	718	...	247	113	369	265	189	198	1
34	school 34:	301	662	755	222	523	403	634	368	400	...	620	608	46	198	672	335	
35	school 35:	818	328	655	356	554	276	811	157	29	...	184	275	797	743	766	93	7
36	school 36:	667	602	544	695	110	781	377	317	424	...	79	866	70	291	100	669	5
37	school 37:	526	309	255	488	675	466	801	12	522	...	460	561	755	514	773	149	1
38	school 38:	796	186	164	628	214	105	243	34	601	...	131	189	707	482	142	584	5
39	school 39:	310	54	325	492	872	454	284	453	443	...	329	539	260	12	297	153	7
40	school 40:	783	595	741	841	388	616	676	146	301	...	569	243	298	307	96	129	5
41	school 41:	360	551	23	84	586	354	794	579	768	...	861	836	294	293	2	120	7
42	school 42:	578	592	366	31	377	859	179	177	725	...	8	310	207	233	121	310	1
43	school 43:	137	446	314	98	714	614	849	46	138	...	832	448	848	487	630	32	3
44	school 44:	179	868	470	620	696	783	397	280	816	...	853	30	514	707	577	494	4
45	school 45:	267	60	615	108	661	816	781	551	734	...	330	800	287	62	802	357	1

	0	1	2	3	4	5	6	7	8	9	...	866	867	868	869	870	871	8
46	school 46:	42	42	709	155	618	500	57	637	283	...	850	515	135	742	13	854	8
47	school 47:	80	558	195	863	51	336	426	829	790	...	214	78	562	338	12	601	2
48	school 48:	183	164	530	302	680	154	156	164	669	...	236	533	232	672	7	687	8
49	school 49:	838	412	817	30	146	823	424	276	805	...	366	127	709	20	32	41	2

50 rows × 876 columns

In [8]:
```
studentdata=studentdata.drop(875)
studentdata
#students 0 thru 874 = 875 students
# rankings 1 thru 50 for 50 schools
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	41	42	43	44	45	46	47	48	49	50
0	student 0:	24	31	41	7	23	10	27	28	31	...	33	24	43	9	7	12	4	49	43	5
1	student 1:	41	21	47	7	49	47	24	42	30	...	27	3	9	3	4	3	27	27	21	24
2	student 2:	42	23	40	19	11	38	32	37	2	...	4	9	8	10	31	15	10	37	2	3
3	student 3:	22	43	28	20	12	48	35	46	30	...	39	33	1	28	25	17	34	13	28	49
4	student 4:	29	22	18	28	5	4	15	40	8	...	30	32	33	4	30	32	1	42	6	30
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
870	student 870:	33	45	8	24	17	4	15	32	12	...	33	34	25	28	1	6	32	19	39	35
871	student 871:	48	39	38	23	37	20	28	18	43	...	41	7	30	29	49	36	32	18	12	10
872	student 872:	42	10	3	11	40	47	49	30	47	...	7	1	41	39	49	15	23	33	48	24
873	student 873:	18	27	11	46	44	43	2	48	4	...	35	17	20	14	20	45	45	37	14	40
874	student 874:	1	29	21	5	38	8	2	32	45	...	27	30	44	15	31	15	11	21	43	42

875 rows × 51 columns

In [9]:
```
studentdata.min(axis=1)
#note that some started their ranking bigger than 0 and had duplicates
```

Out[9]:
```
0       0
1       0
2       2
3       1
4       1
       ..
870     1
871     0
872     0
873     0
874     0
Length: 875, dtype: int64
```

In [10]:
```
pia= pd.DataFrame(schooldata.mean()*(1/875))
pia=pia.reset_index(drop=True)
pia
```

Out[10]:

	0
0	0.509371
1	0.498469
2	0.509257
3	0.506057
4	0.520114
...	...
870	0.477211
871	0.456297
872	0.474629
873	0.520046
874	0.495017

875 rows × 1 columns

In [11]:
```
pib= pd.DataFrame(studentdata.mean(axis=1)*(1/50))
pib
```

Out[11]:

	0
0	0.5528
1	0.4856
2	0.4904
3	0.5508
4	0.5316
...	...
870	0.4812
871	0.4612
872	0.5104
873	0.5300
874	0.4296

875 rows × 1 columns

In [12]:
```
comp=pia+pib- (pia*pib)
comp
```

Out[12]:

	0
0	0.780591
1	0.742012
2	0.749917
3	0.778121
4	0.775222
...	...
870	0.728777
871	0.707053
872	0.742778
873	0.774421
874	0.711958

875 rows × 1 columns

In [13]:
```
#for this algorithm we need all schools to have the same cap so \
#students has to be multiple of schools for locality caps.. so we will
#truncate students at 850 aka student 849 and set caps =17
studentdata=studentdata.drop(studentdata.index[850:875])
studentdata
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9	...	41	42	43	44	45	46	47	48	49	50
0	student 0:	24	31	41	7	23	10	27	28	31	...	33	24	43	9	7	12	4	49	43	5
1	student 1:	41	21	47	7	49	47	24	42	30	...	27	3	9	3	4	3	27	27	21	24
2	student 2:	42	23	40	19	11	38	32	37	2	...	4	9	8	10	31	15	10	37	2	3
3	student 3:	22	43	28	20	12	48	35	46	30	...	39	33	1	28	25	17	34	13	28	49
4	student 4:	29	22	18	28	5	4	15	40	8	...	30	32	33	4	30	32	1	42	6	30
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
845	student 845:	49	9	30	5	23	22	36	33	29	...	7	8	16	24	20	48	37	37	46	21
846	student 846:	35	34	43	1	30	12	46	16	37	...	19	10	30	1	2	27	11	33	32	20
847	student 847:	13	40	24	45	18	47	12	47	49	...	11	48	15	19	6	7	25	20	2	40
848	student 848:	26	37	24	34	17	17	36	21	3	...	14	22	6	40	21	13	20	44	18	11
849	student 849:	42	33	32	38	32	44	16	1	39	...	22	12	8	46	12	31	10	48	9	7

850 rows × 51 columns

In [14]:
```
##similarly we need to truncate school data
schooldata=schooldata.drop(schooldata.iloc[:, 851:876], axis=1)
```

In [15]:
```
schooldata
```

Out[15]:

	0	1	2	3	4	5	6	7	8	9	...	841	842	843	844	845	846	8
0	school 0:	314	437	195	270	100	388	768	95	193	...	105	148	614	86	822	736	5
1	school 1:	534	334	767	801	196	738	812	63	434	...	780	499	511	421	275	223	7
2	school 2:	657	348	674	622	708	331	283	55	466	...	183	729	701	669	322	426	4
3	school 3:	567	607	454	583	320	60	627	177	201	...	454	767	780	207	248	564	
4	school 4:	845	814	265	477	84	855	85	408	298	...	195	543	596	414	255	194	7
5	school 5:	851	478	256	220	66	461	395	346	402	...	204	47	409	77	46	559	4
6	school 6:	70	429	257	863	597	340	843	421	364	...	811	386	319	385	570	712	5
7	school 7:	586	402	248	204	186	395	738	820	698	...	439	448	346	184	652	447	2
8	school 8:	53	584	807	36	294	662	602	649	235	...	601	698	230	868	392	135	2
9	school 9:	435	497	701	766	788	118	620	572	200	...	84	707	351	672	399	742	5
10	school 10:	17	595	125	324	837	660	128	7	513	...	592	64	147	528	698	683	
11	school 11:	841	843	419	749	435	616	621	46	178	...	796	148	174	498	307	387	2
12	school 12:	283	84	537	557	742	592	453	862	46	...	182	775	404	55	324	332	3
13	school 13:	805	95	670	431	706	695	571	27	797	...	521	771	97	791	538	239	
14	school 14:	311	370	533	866	225	192	452	381	319	...	52	27	182	330	155	837	2
15	school 15:	372	304	332	73	299	629	801	486	201	...	430	735	349	323	406	366	7
16	school 16:	341	147	778	702	755	138	39	300	377	...	573	872	152	854	558	521	5
17	school 17:	676	678	491	292	99	813	615	671	838	...	280	226	850	336	500	764	7
18	school 18:	281	601	466	203	336	624	290	298	394	...	628	467	514	104	149	658	4
19	school 19:	631	340	265	815	210	202	146	130	404	...	40	790	75	499	819	293	6
20	school 20:	379	609	667	535	756	392	192	425	338	...	516	752	313	379	429	707	
21	school 21:	16	321	276	114	873	751	275	20	317	...	451	486	786	386	442	169	8
22	school 22:	874	55	102	566	500	667	713	382	291	...	659	348	266	176	556	50	1

	0	1	2	3	4	5	6	7	8	9	...	841	842	843	844	845	846	8
23	school 23:	571	310	584	190	765	678	534	401	76	...	139	478	870	297	407	250	3
24	school 24:	542	870	220	223	296	426	505	792	238	...	473	642	192	68	393	856	7
25	school 25:	811	767	550	77	135	41	514	54	667	...	521	548	726	527	810	873	5
26	school 26:	32	724	32	847	662	171	410	773	421	...	348	0	695	295	458	250	1
27	school 27:	335	157	376	509	848	788	268	7	310	...	304	690	18	110	370	384	6
28	school 28:	448	353	646	63	245	666	93	281	70	...	292	352	780	47	649	307	8
29	school 29:	750	226	238	298	173	699	378	786	404	...	493	265	831	246	382	210	5
30	school 30:	199	164	176	716	95	724	859	738	22	...	241	463	108	842	561	525	1
31	school 31:	56	513	548	789	137	357	54	624	372	...	310	169	400	263	90	137	1
32	school 32:	873	467	523	482	769	600	31	718	342	...	357	403	802	174	558	729	6
33	school 33:	64	856	624	864	653	708	285	6	718	...	457	697	353	766	608	417	1
34	school 34:	301	662	755	222	523	403	634	368	400	...	603	220	842	237	264	755	8
35	school 35:	818	328	655	356	554	276	811	157	29	...	81	806	623	482	762	873	7
36	school 36:	667	602	544	695	110	781	377	317	424	...	575	795	100	32	447	730	
37	school 37:	526	309	255	488	675	466	801	12	522	...	254	394	228	721	95	483	6
38	school 38:	796	186	164	628	214	105	243	34	601	...	479	226	305	683	621	785	4
39	school 39:	310	54	325	492	872	454	284	453	443	...	709	133	57	581	716	688	7
40	school 40:	783	595	741	841	388	616	676	146	301	...	655	117	867	456	223	708	7
41	school 41:	360	551	23	84	586	354	794	579	768	...	80	89	521	151	109	832	5
42	school 42:	578	592	366	31	377	859	179	177	725	...	666	148	607	705	69	218	4
43	school 43:	137	446	314	98	714	614	849	46	138	...	627	114	856	517	324	695	8
44	school 44:	179	868	470	620	696	783	397	280	816	...	853	851	697	83	190	228	1
45	school 45:	267	60	615	108	661	816	781	551	734	...	545	442	365	832	32	598	

	0	1	2	3	4	5	6	7	8	9	...	841	842	843	844	845	846	8
46	school 46:	42	42	709	155	618	500	57	637	283	...	569	635	469	387	819	72	
47	school 47:	80	558	195	863	51	336	426	829	790	...	201	44	57	742	526	784	5
48	school 48:	183	164	530	302	680	154	156	164	669	...	778	371	708	300	130	52	2
49	school 49:	838	412	817	30	146	823	424	276	805	...	435	514	423	332	122	599	2

50 rows × 851 columns

In [33]:
```
pia=pia.drop(pia.index[850:875])
pib=pib.drop(pib.index[850:875])
comp= comp.drop(comp.index[850:875])
pia
```

Out[33]:

	0
0	0.509371
1	0.498469
2	0.509257
3	0.506057
4	0.520114
...	...
845	0.566446
846	0.507771
847	0.524571
848	0.506286
849	0.542674

850 rows × 1 columns

In [35]:
```
comp
```

Out[35]:

	0
0	0.780591
1	0.742012
2	0.749917
3	0.778121
4	0.775222
...	...
845	0.781662
846	0.755067
847	0.785867
848	0.733592
849	0.760544

850 rows × 1 columns

In [37]:
```
pia100=pia.drop(pia.index[100:850])
pib100=pib.drop(pib.index[100:850])
comp100= comp.drop(comp.index[100:850])
pia100
```

Out[37]:

	0
0	0.509371
1	0.498469
2	0.509257
3	0.506057
4	0.520114
...	...
95	0.549394
96	0.413211
97	0.536709
98	0.508800
99	0.489006

100 rows × 1 columns

In [331]:
```
len(studentdata)
```

Out[331]: 850

In [ ]:
```
#consider localities= school 0 thru school 49.
#num_localities= 50
#cap each at ceiling 850/50=17
#prof 1= honorsA= 425
#prof 2= honorsB
#competency= comp= pia or pib: prob student i gets into A or B
```

In [38]:
```
seed(0)
```

In [39]:
```
50*17
```

Out[39]:
```
850
```

In [40]:
```
num_professions = 2  # This is a constant; changing it requires
                     # further code modifications
num_agents = 100 #number of students-- can be changed to 850 but will ta
ke my lifetime to run
prof1 = 50 #number of spots in honors A
prof2 = num_agents - prof1 #number of spots in honors B
professions = [0] * prof1 + [1] * prof2
num_localities = 10  # Fix localities to 50 localities with cap 17 each
 for 850
locality_caps = [10] * 10 #for 850 students =[17] * 50
random_samples = 300 #decreased to improve computation time

def _distribute_jobs(prof1_jobs, prof2_jobs):
    job_numbers = [(0, 0)] * num_localities
    for _ in range(prof1_jobs):
        l = randrange(num_localities)
        prof1, prof2 = job_numbers[l]
        job_numbers[l] = (prof1 + 1, prof2)
    for _ in range(prof2_jobs):
        l = randrange(num_localities)
        prof1, prof2 = job_numbers[l]
        job_numbers[l] = (prof1, prof2 + 1)

    return job_numbers

def test_correction(prof1_jobs, prof2_jobs):
    job_numbers = _distribute_jobs(prof1_jobs, prof2_jobs)
    qualification_probabilities =\
    [[comp.iloc[_,0]]* num_localities for _ in range(num_agents)]
    #include comp instead of random()
    correction_functions = []
    for p1, p2 in job_numbers:
        # The default parameters in the lambdas are never used, but are
        # a way of getting Python's peculiar binding behavior to work.
        # See https://docs.python.org/3/faq/programming.html#why-do-
        # lambdas-defined-in-a-loop-with-different-values-all-return-
        # the-same-result for more information.
        correction_functions.append((lambda x, P1=p1: min(x, P1),
                                     lambda x, P2=p2: min(x, P2)))
    model = RetroactiveCorrectionModel(num_agents, locality_caps,
                                       num_professions, professions,
                                       qualification_probabilities,
                                       correction_functions,
                                       random_samples)
    return model

def test_interview(prof1_jobs, prof2_jobs):
    job_numbers = _distribute_jobs(prof1_jobs, prof2_jobs)
    #makes sense to keep compatability prob random
    compatibility_probabilities = [random() for _ in range(num_agents)]
    model = InterviewModel(num_agents, locality_caps, num_professions,
                           professions, job_numbers,
                           compatibility_probabilities, random_samples)
    return model
```

```
#competency should be the average of ratings for each student given by t
he school
def test_coordination(prof1_jobs, prof2_jobs):
    job_numbers = _distribute_jobs(prof1_jobs, prof2_jobs)
    locality_num_jobs = [prof1 + prof2 for prof1, prof2 in job_numbers]
    compatibility_probabilities = []
    for _ in range(prof1):
        competency = pia.iloc[_,0]
        compatibility_probabilities.append(
            [[competency] * p1 + [0.] * p2 for p1, p2 in job_numbers])
    for _ in range(prof2):
        competency = pib.iloc[_,0]
        compatibility_probabilities.append(
            [[0.] * p1 + [competency] * p2 for p1, p2 in job_numbers])
    model = CoordinationModel(num_agents, locality_caps,
                              locality_num_jobs,
                              compatibility_probabilities,
                              random_samples)
    return model

settings = {"correction": test_correction, "interview": test_interview,
            "coordination": test_coordination}
```

In [20]:
```
settings["correction"]
```

Out[20]:
```
<function __main__.test_correction(prof1_jobs, prof2_jobs)>
```

In [345]:
```
len(professions)
```

Out[345]:
```
850
```

In [41]:
```
data = []

def sample(setting, prof1_jobs, prof2_jobs):
    m = settings[setting](prof1_jobs, prof2_jobs)
    additive = additive_optimization(m)[1]
    greedy = greedy_algorithm(m)[1]
    datum = {}
    datum["jobs in profession 1"] = prof1_jobs
    datum["jobs in profession 2"] = prof2_jobs
    datum["additive"] = additive
    datum["greedy"] = greedy
    if additive > 0.0005:
        datum["greedy / additive"] = greedy / additive
    else:
        datum["greedy / additive"] = None
    datum["model"] = setting
    data.append(datum)
    return datum
```

In [22]:
```
print(data)
```

```
[]
```

In [42]:
```
sample("correction", 25, 25)
```

Out[42]:
```
{'jobs in profession 1': 25,
 'jobs in profession 2': 25,
 'additive': 44.71666666666666,
 'greedy': 48.436666666666675,
 'greedy / additive': 1.0831904584420429,
 'model': 'correction'}
```

In [43]:
```
from datetime import datetime
for _ in range(5):
    for prof1_jobs in [25, 50, 75]:
        for prof2_jobs in [25, 50, 75]:
            for k in settings:
                sample(k, prof1_jobs, prof2_jobs)
                print(datetime.now(), k, prof1_jobs, prof2_jobs)
from pickle import dumps
dumps(data)
```

```
2021-03-14 22:46:57.532911 correction 25 25
2021-03-14 22:47:05.138036 interview 25 25
2021-03-14 22:49:33.447855 coordination 25 25
2021-03-14 22:49:38.602641 correction 25 50
2021-03-14 22:49:46.332566 interview 25 50
2021-03-14 22:52:29.659912 coordination 25 50
2021-03-14 22:52:34.727359 correction 25 75
2021-03-14 22:52:42.650488 interview 25 75
2021-03-14 22:55:44.077652 coordination 25 75
2021-03-14 22:55:49.021129 correction 50 25
2021-03-14 22:55:56.749463 interview 50 25
2021-03-14 22:58:46.494033 coordination 50 25
2021-03-14 22:58:51.441334 correction 50 50
2021-03-14 22:58:59.283400 interview 50 50
2021-03-14 23:02:02.183108 coordination 50 50
2021-03-14 23:02:07.285082 correction 50 75
2021-03-14 23:02:15.992288 interview 50 75
2021-03-14 23:35:29.778168 coordination 50 75
2021-03-14 23:35:34.854039 correction 75 25
2021-03-14 23:35:42.536549 interview 75 25
2021-03-14 23:38:40.088765 coordination 75 25
2021-03-14 23:38:45.187994 correction 75 50
2021-03-14 23:38:53.108116 interview 75 50
2021-03-14 23:42:09.402540 coordination 75 50
2021-03-14 23:42:14.506426 correction 75 75
2021-03-14 23:42:22.601104 interview 75 75
2021-03-14 23:45:42.364250 coordination 75 75
2021-03-14 23:45:47.599441 correction 25 25
2021-03-14 23:45:55.728648 interview 25 25
2021-03-14 23:48:25.570661 coordination 25 25
2021-03-14 23:48:30.499168 correction 25 50
2021-03-14 23:48:38.599116 interview 25 50
2021-03-14 23:51:26.757979 coordination 25 50
2021-03-14 23:51:31.832393 correction 25 75
2021-03-14 23:51:39.960707 interview 25 75
2021-03-14 23:54:35.718039 coordination 25 75
2021-03-14 23:54:41.850033 correction 50 25
2021-03-14 23:54:51.331986 interview 50 25
2021-03-14 23:57:38.196100 coordination 50 25
2021-03-14 23:57:43.194096 correction 50 50
2021-03-14 23:57:51.045643 interview 50 50
2021-03-15 00:00:53.845410 coordination 50 50
2021-03-15 00:01:00.154215 correction 50 75
2021-03-15 00:01:10.065321 interview 50 75
2021-03-15 00:04:35.764032 coordination 50 75
2021-03-15 00:04:40.902478 correction 75 25
2021-03-15 00:04:48.684320 interview 75 25
2021-03-15 00:07:51.798411 coordination 75 25
2021-03-15 00:07:56.964911 correction 75 50
2021-03-15 00:08:05.565474 interview 75 50
2021-03-15 00:11:16.950497 coordination 75 50
2021-03-15 00:11:21.930429 correction 75 75
2021-03-15 00:11:29.838984 interview 75 75
2021-03-15 00:14:50.162409 coordination 75 75
2021-03-15 00:14:55.104212 correction 25 25
2021-03-15 00:15:03.209214 interview 25 25
2021-03-15 00:17:35.846944 coordination 25 25
```

```
2021-03-15 00:17:41.089925 correction 25 50
2021-03-15 00:17:49.370807 interview 25 50
2021-03-15 00:20:43.067050 coordination 25 50
2021-03-15 00:20:48.302480 correction 25 75
2021-03-15 00:20:56.486537 interview 25 75
2021-03-15 00:24:11.707310 coordination 25 75
2021-03-15 00:24:16.985988 correction 50 25
2021-03-15 00:24:25.102003 interview 50 25
2021-03-15 00:27:16.830730 coordination 50 25
2021-03-15 00:27:22.171357 correction 50 50
2021-03-15 00:27:30.575528 interview 50 50
2021-03-15 00:30:36.617861 coordination 50 50
2021-03-15 00:30:41.657253 correction 50 75
2021-03-15 00:30:49.817202 interview 50 75
2021-03-15 00:34:03.491065 coordination 50 75
2021-03-15 00:34:08.581248 correction 75 25
2021-03-15 00:34:16.259668 interview 75 25
2021-03-15 00:37:17.246497 coordination 75 25
2021-03-15 00:37:22.360123 correction 75 50
2021-03-15 00:37:30.369452 interview 75 50
2021-03-15 00:40:43.010805 coordination 75 50
2021-03-15 00:40:48.153580 correction 75 75
2021-03-15 00:40:56.389327 interview 75 75
2021-03-15 00:44:27.162363 coordination 75 75
2021-03-15 00:44:32.707596 correction 25 25
2021-03-15 00:44:40.904471 interview 25 25
2021-03-15 00:47:15.894942 coordination 25 25
2021-03-15 00:47:21.232784 correction 25 50
2021-03-15 00:47:29.374434 interview 25 50
2021-03-15 00:50:20.854842 coordination 25 50
2021-03-15 00:50:25.989954 correction 25 75
2021-03-15 00:50:34.332646 interview 25 75
2021-03-15 00:53:39.328677 coordination 25 75
2021-03-15 00:53:44.725246 correction 50 25
2021-03-15 00:53:52.868108 interview 50 25
2021-03-15 00:56:45.890989 coordination 50 25
2021-03-15 00:56:51.069272 correction 50 50
2021-03-15 00:56:58.685751 interview 50 50
2021-03-15 00:59:56.415664 coordination 50 50
2021-03-15 01:00:01.394781 correction 50 75
2021-03-15 01:00:09.153686 interview 50 75
2021-03-15 01:03:20.654972 coordination 50 75
2021-03-15 01:03:25.974033 correction 75 25
2021-03-15 01:03:34.100447 interview 75 25
2021-03-15 01:06:52.396444 coordination 75 25
2021-03-15 01:06:58.399407 correction 75 50
2021-03-15 01:07:07.895342 interview 75 50
2021-03-15 01:10:29.683870 coordination 75 50
2021-03-15 01:10:34.917482 correction 75 75
2021-03-15 01:10:43.274783 interview 75 75
2021-03-15 01:14:09.757949 coordination 75 75
2021-03-15 01:14:14.936829 correction 25 25
2021-03-15 01:14:23.246044 interview 25 25
2021-03-15 01:16:59.266226 coordination 25 25
2021-03-15 01:17:04.279188 correction 25 50
2021-03-15 01:17:12.190773 interview 25 50
2021-03-15 01:19:58.461447 coordination 25 50
```

```
2021-03-15 01:20:03.645557 correction 25 75
2021-03-15 01:20:11.997041 interview 25 75
2021-03-15 01:23:16.069866 coordination 25 75
2021-03-15 01:23:21.483784 correction 50 25
2021-03-15 01:23:29.953324 interview 50 25
2021-03-15 01:26:22.856440 coordination 50 25
2021-03-15 01:26:28.283732 correction 50 50
2021-03-15 01:26:36.991818 interview 50 50
2021-03-15 01:29:41.654732 coordination 50 50
2021-03-15 01:29:46.907545 correction 50 75
2021-03-15 01:29:55.302733 interview 50 75
2021-03-15 01:33:12.094339 coordination 50 75
2021-03-15 01:33:17.808723 correction 75 25
2021-03-15 01:33:26.383979 interview 75 25
2021-03-15 01:36:36.210092 coordination 75 25
2021-03-15 01:36:41.763292 correction 75 50
2021-03-15 01:36:50.383449 interview 75 50
2021-03-15 01:40:11.287209 coordination 75 50
2021-03-15 01:40:16.488947 correction 75 75
2021-03-15 01:40:24.601146 interview 75 75
2021-03-15 01:43:40.605954 coordination 75 75
```

Out[43]:
b'\x80\x03]q\x00(}q\x01(X\x14\x00\x00\x00jobs in profession 1q\x02K\x19
X\x14\x00\x00\x00jobs in profession 2q\x03K\x19X\x08\x00\x00\x00additiv
eq\x04G@F[\xbb\xbb\xbb\xbb\xbbX\x06\x00\x00\x00greedyq\x05G@H7\xe4\xb1~
K\x19X\x11\x00\x00\x00greedy / additiveq\x06G?\xf1T\xbf\x84\xa5\x90\xad
X\x05\x00\x00\x00modelq\x07X\n\x00\x00\x00correctionq\x08u}q\t(h\x02K\x
19h\x03K\x19h\x04G@F\xda\xe1G\xae\x14zh\x05G@G\xf4z\xe1G\xae\x15h\x06G?
\xf0\xc5#s\x89\xf0\xe2h\x07h\x08u}q\n(h\x02K\x19h\x03K\x19h\x04G@@\xe8
\x88\x88\x88\x88\x88h\x05G@H\x05\x1e\xb8Q\xeb\x84h\x06G?\xf6\xba\xc8h\x
c5\xa2.h\x07X\t\x00\x00\x00interviewq\x0bu}q\x0c(h\x02K\x19h\x03K\x19h
\x04G@B\x11~K\x17\xe4\xb1h\x05G@H\x15UUUUVh\x06G?\xf5S\x91\xf5_\xa9\x1b
h\x07X\x0c\x00\x00\x00coordinationq\ru}q\x0e(h\x02K\x19h\x03K2h\x04G@J
\x1f\x92\xc5\xf9,`h\x05G@N]p\xa3\xd7\n=h\x06G?\xf2\x99\x12!\x1ft\xf9h\x
07h\x08u}q\x0f(h\x02K\x19h\x03K2h\x04G@G\x8e\x14z\xe1G\xaeh\x05G@N\xe0
m:\x06\xd3\xa1h\x06G?\xf4\xf9+\xd1\x1f5\xb5h\x07h\x0bu}q\x10(h\x02K\x19
h\x03K2h\x04G@E\xfa\xe1G\xae\x14{h\x05G@Qv\xd3\xa0m:\x08h\x06G?\xf91\xf
0\x15\x1e\xab\x9bh\x07h\ru}q\x11(h\x02K\x19h\x03KKh\x04G@M\x86ffffh\x0
5G@M\xb7\xe4\xb1~K\x19h\x06G?\xf0\x1a\xd24fAih\x07h\x08u}q\x12(h\x02K\x
19h\x03KKh\x04G@KZ\x06\xd3\xa0m:h\x05G@P\x8dp\xa3\xd7\n=h\x06G?\xf3]\x9
4\x85W\xbe\xfbh\x07h\x0bu}q\x13(h\x02K\x19h\x03KKh\x04G@O\xe5\x1e\xb8Q
\xeb\x86h\x05G@P\x85\x1e\xb8Q\xeb\x85h\x06G?\xf0\x93\n\xe0,\xf2\x8eh\x0
7h\ru}q\x14(h\x02K2h\x03K\x19h\x04G@J\x82\x8f\\(\xf5\xc2h\x05G@L\x80\xd
at\r\xa7@h\x06G?\xf13\xfcD\x1c\xb6sh\x07h\x08u}q\x15(h\x02K2h\x03K\x19h
\x04G@I`\x00\x00\x00\x00\x00h\x05G@P\x10\xdat\r\xa7Ah\x06G?\xf4B\xa7\x0
8\xe5\x14\x7fh\x07h\x0bu}q\x16(h\x02K2h\x03K\x19h\x04G@I.\x81\xb4\xe8\x
1bOh\x05G@Q\x1f%\x8b\xf2X\xbfh\x06G?\xf5\xc1\xf3\xe6\xfd\xc7\xc7h\x07h
\ru}q\x17(h\x02K2h\x03K2h\x04G@M\xdd\xdd\xdd\xdd\xdd\xdeh\x05G@Q\x1f%\x
8b\xf2X\xc0h\x06G?\xf2X:\x83\xa8:\x84h\x07h\x08u}q\x18(h\x02K2h\x03K2h
\x04G@O:t\r\xa7@\xdah\x05G@Q\x9e\xee\xee\xee\xee\xefh\x06G?\xf2\x0ef\x8
d\x92\xf1\x88h\x07h\x0bu}q\x19(h\x02K2h\x03K2h\x04G@QhQ\xeb\x85\x1e\xb9
h\x05G@T\xb0\xdat\r\xa7Bh\x06G?\xf3\x04\x92:\xcc\xfb<h\x07h\ru}q\x1a(h
\x02K2h\x03KKh\x04G@QW\xae\x14z\xe1Gh\x05G@R(\xf5\xc2\x8f\\)h\x06G?\xf0
\xc1\x14Uw\xf96h\x07h\x08u}q\x1b(h\x02K2h\x03KKh\x04G@R\xdf\xc9b\xfc\x9
60h\x05G@S\xd8Q\xeb\x85\x1e\xb8h\x06G?\xf0\xd2\xaf\xc2\xb1c]h\x07h\x0b
u}q\x1c(h\x02K2h\x03KKh\x04G@R\xf2X\xbf%\x8b\xf2h\x05G@U\x0f\\(\xf5\xc2
\x90h\x06G?\xf1\xc8\xdfdX\xa2\xddh\x07h\ru}q\x1d(h\x02KKh\x03K\x19h\x04
G@M\xa4DDDDDh\x05G@M\x88\x88\x88\x88\x88\x88h\x06G?\xef\xe2\x0ff\xd8\xa
1\xcah\x07h\x08u}q\x1e(h\x02KKh\x03K\x19h\x04G@K&ffffgh\x05G@Q\x87\xe4
\xb1~K\x17h\x06G?\xf4\xa9\x99\x18\xdc\x83\xe0h\x07h\x0bu}q\x1f(h\x02KKh
\x03K\x19h\x04G@Q\x1d\xa7@\xdat\x0eh\x05G@P\xd1\xb4\xe8\x1bN\x82h\x06G?
\xefr\x02Q}\x80>h\x07h\ru}q (h\x02KKh\x03K2h\x04G@Ql\x96/\xc9b\xfch\x05
G@Q\xef\x92\xc5\xf9,`h\x06G?\xf0xG\xcb\xcb(-h\x07h\x08u}q!(h\x02KKh\x03
K2h\x04G@R\x92X\xbf%\x8b\xf3h\x05G@S\xbbN\x81\xb4\xe8\x1bh\x06G?\xf0\xf
f\xd6\xd4\x9a\x1c*h\x07h\x0bu}q"(h\x02KKh\x03K2h\x04G@S\xd8\xbf%\x8b\xf
2Yh\x05G@T6/\xc9b\xfc\x96h\x06G?\xf0KT[\xd7\x11Hh\x07h\ru}q#(h\x02KKh\x
03KKh\x04G@Rn\x14z\xe1G\xaeh\x05G@R[\x85\x1e\xb8Q\xech\x06G?\xef\xdf\xc
67H\xa5\x94h\x07h\x08u}q$(h\x02KKh\x03KKh\x04G@T\xdd\x03i\xd06\x9ch\x05
G@T\x7f\xff\xff\xff\xff\xffh\x06G?\xefqVB\xab6Gh\x07h\x0bu}q%(h\x02KKh
\x03KKh\x04G@V\xef\\(\xf5\xc2\x8fh\x05G@W\xd5\xf9,_\x92\xc6h\x06G?\xf0
\xa0\xe1\x9e\xed\x90kh\x07h\ru}q&(h\x02K\x19h\x03K\x19h\x04G@F\xa7@\xda
t\r\xa6h\x05G@GY,_\x92\xc5\xfah\x06G?\xf0}\xaa\x1d;7Nh\x07h\x08u}q\'(h
\x02K\x19h\x03K\x19h\x04G@?1~K\x17\xe4\xb2h\x05G@Ga\xb4\xe8\x1bN\x82h\x
06G?\xf7\xfc\x7f\x96\xf3\xb0\x8fh\x07h\x0bu}q((h\x02K\x19h\x03K\x19h\x0
4G@@\xfat\r\xa7@\xdah\x05G@H2X\xbf%\x8b\xf3h\x06G?\xf6\xcdi\xc5\xbf\x9d
\xe6h\x07h\ru}q)(h\x02K\x19h\x03K2h\x04G@J\x14\xe8\x1bN\x81\xb5h\x05G@L
aG\xae\x14z\xe0h\x06G?\xf1h\xf11\xf9\xf2|h\x07h\x08u}q*(h\x02K\x19h\x03
K2h\x04G@Gtz\xe1G\xae\x15h\x05G@N\x10\xa3\xd7\n=qh\x06G?\xf4\x82Q\xbf\x
ee\x89jh\x07h\x0bu}q+(h\x02K\x19h\x03K2h\x04G@HW\xe4\xb1~K\x18h\x05G@P

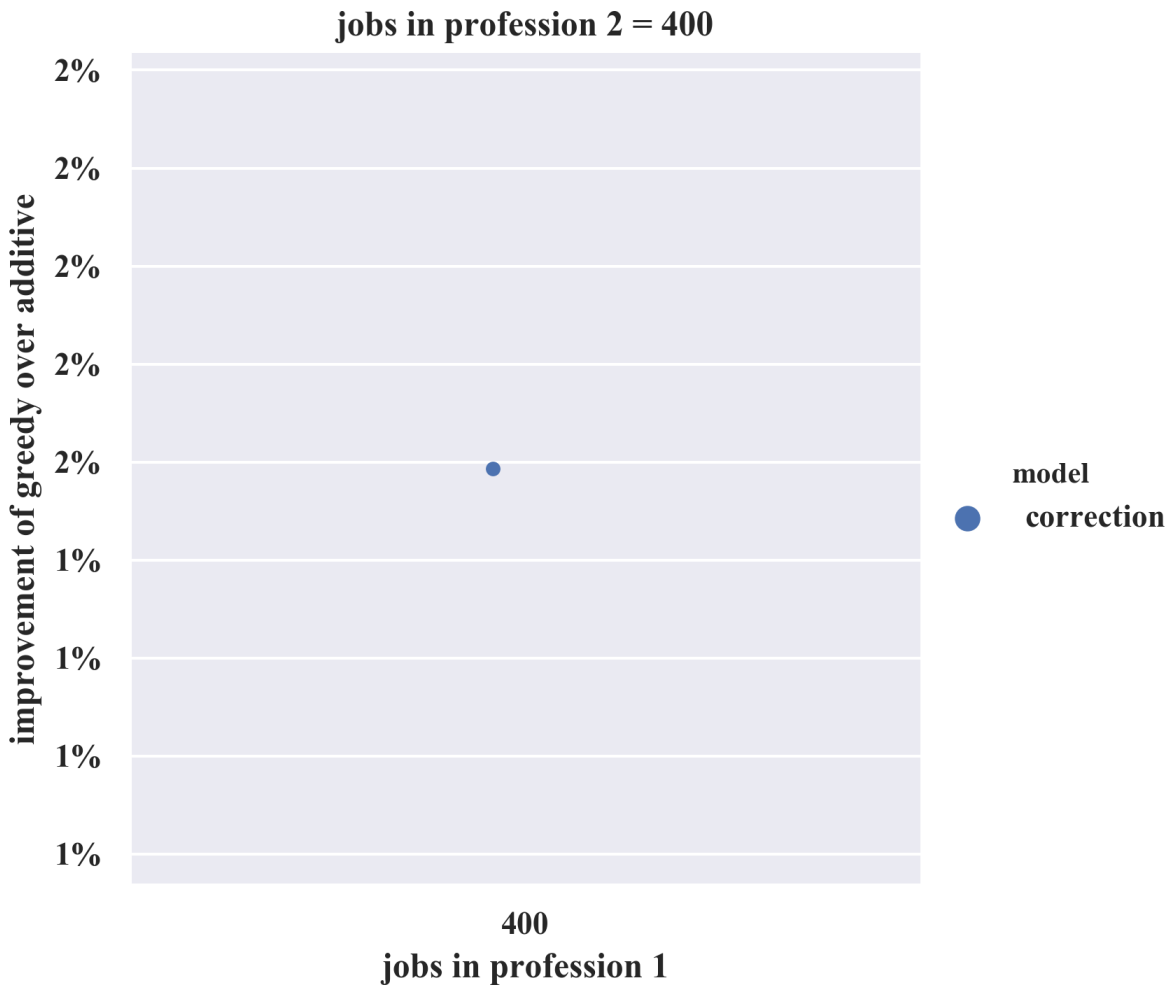
\xa6\x9d\x03i\xd07h\x06G?\xf5\xe3Sd\xc7\xce\xedh\x07h\ru}q,(h\x02K\x19h
\x03KKh\x04G@M\xe5\x8b\xf2X\xbf\'h\x05G@M\x98\xbf%\x8b\xf2Yh\x06G?\xef
\xad\xcb\xfe\xbe~\x8eh\x07h\x08u}q-(h\x02K\x19h\x03KKh\x04G@L\x91\x11\x
11\x11\x11\x12h\x05G@P\xd1G\xae\x14z\xe1h\x06G?\xf2\xd6\xb7\xa6\x9f\xc0
\xe8h\x07h\x0bu}q.(h\x02K\x19h\x03KKh\x04G@O\xe5\x1e\xb8Q\xeb\x86h\x05G
@P\xcf\x92\xc5\xf9,`h\x06G?\xf0\xdd\xbd\xac\xfb\x1e\x08h\x07h\ru}q/(h\x
02K2h\x03K\x19h\x04G@I\xe5\x1e\xb8Q\xeb\x86h\x05G@MQ~K\x17\xe4\xb0h\x06
G?\xf2\x1d~S\xdd\xed\x0ch\x07h\x08u}q0(h\x02K2h\x03K\x19h\x04G@H;N\x81
\xb4\xe8\x1bh\x05G@N\xee\x81\xb4\xe8\x1bOh\x06G?\xf4l\x88\xd9\x11\xb2$h
\x07h\x0bu}q1(h\x02K2h\x03K\x19h\x04G@LI\xd06\x9d\x03jh\x05G@Q\xa4\xb1~
K\x17\xe4h\x06G?\xf3\xf5RF\xccn\x95h\x07h\ru}q2(h\x02K2h\x03K2h\x04G@N
\xda\x06\xd3\xa0m:h\x05G@P\xf2\xc5\xf9,_\x94h\x06G?\xf1\x94Dm\xce\xa0\x
d8h\x07h\x08u}q3(h\x02K2h\x03K2h\x04G@OE\x8b\xf2X\xbf&h\x05G@R\x85\x1e
\xb8Q\xeb\x85h\x06G?\xf2\xf3\x8b\x0br|Yh\x07h\x0bu}q4(h\x02K2h\x03K2h\x
04G@Q/\\(\xf5\xc2\x90h\x05G@U,(\xf5\xc2\x8f]h\x06G?\xf3\xb6i\x05\x96\x1
5\x9ah\x07h\ru}q5(h\x02K2h\x03KKh\x04G@Qb\x8f\\(\xf5\xc2h\x05G@Q\xfb\x1
7\xe4\xb1~Kh\x06G?\xf0\x8ca\xad\x96R8h\x07h\x08u}q6(h\x02K2h\x03KKh\x04
G@R\xc5\xf9,_\x92\xc6h\x05G@S\x1b\x85\x1e\xb8Q\xech\x06G?\xf0H\xe8\xb
a.\x8b\xa3h\x07h\x0bu}q7(h\x02K2h\x03KKh\x04G@R\xf4\xb1~K\x17\xe6h\x05G
@U\xc2X\xbf%\x8b\xf2h\x06G?\xf2]\xbflX0)h\x07h\ru}q8(h\x02KKh\x03K\x19h
\x04G@L.\x14z\xe1G\xaeh\x05G@M\xdc\x96/\xc9b\xfbh\x06G?\xf0\xf4n\xb5\x9
1iph\x07h\x08u}q9(h\x02KKh\x03K\x19h\x04G@L\x12X\xbf%\x8b\xf2h\x05G@P\x
9eK\x17\xe4\xb1}h\x06G?\xf2\xf1\xa3\x02\xe7.\x01h\x07h\x0bu}q:(h\x02KKh
\x03K\x19h\x04G@K\xdd\xdd\xdd\xdd\xdd\xdeh\x05G@P\x8c\xcc\xcc\xcc\xcc\x
cdh\x06G?\xf3\x019\x91\xc2\xc1\x88h\x07h\ru}q;(h\x02KKh\x03K2h\x04G@P\x
89\xd06\x9d\x03kh\x05G@Q\xec\xcc\xcc\xcc\xcc\xcch\x06G?\xf1Wn~\xe0\xb4
\xcah\x07h\x08u}q<(h\x02KKh\x03K2h\x04G@Q\xc7wwwwxh\x05G@S\x9f\xc9b\xfc
\x961h\x06G?\xf1\xa9\x0e\x06\xe30\xb5h\x07h\x0bu}q=(h\x02KKh\x03K2h\x04
G@S\x9e\x81\xb4\xe8\x1bOh\x05G@U\x85UUUUUh\x06G?\xf1\x8d\x05r\x9e\xb0\x
eah\x07h\ru}q>(h\x02KKh\x03KKh\x04G@Q\x14z\xe1G\xae\x15h\x05G@R\x89\xd0
6\x9d\x03ih\x06G?\xf1]\xba\r\xfd3\xc1h\x07h\x08u}q?(h\x02KKh\x03KKh\x04
G@T\xaa=p\xa3\xd7\x0bh\x05G@T\xfe\xb8Q\xeb\x85 h\x06G?\xf0Ah\xbd\xc9\x1
e\xe9h\x07h\x0bu}q@(h\x02KKh\x03KKh\x04G@U7wwwwwh\x05G@W3\xd7\n=p\xa4h
\x06G?\xf1\x7f`h@\x07\xb9h\x07h\ru}qA(h\x02K\x19h\x03K\x19h\x04G@D\x18Q
\xeb\x85\x1e\xb9h\x05G@GN\x14z\xe1G\xaeh\x06G?\xf2\x8eL\xd1\xb1$\x7fh\x
07h\x08u}qB(h\x02K\x19h\x03K\x19h\x04G@Dffffffh\x05G@G\xe1\xb4\xe8\x1bN
\x83h\x06G?\xf2\xbb\x10e\xbb\x10gh\x07h\x0bu}qC(h\x02K\x19h\x03K\x19h\x
04G@C \x00\x00\x00\x00\x00h\x05G@H.\xee\xee\xee\xee\xefh\x06G?\xf4;W\xc
9\x90\xad\x1fh\x07h\ru}qD(h\x02K\x19h\x03K2h\x04G@L\x00\xdat\r\xa7@h\x0
5G@M\xc1\xb4\xe8\x1bN\x82h\x06G?\xf1\x00u\x03\xbf\x85#h\x07h\x08u}qE(h
\x02K\x19h\x03K2h\x04G@Fp6\x9d\x03i\xd1h\x05G@M\xce\x14z\xe1G\xaeh\x06
G?\xf5@\xbf\x10B\x055h\x07h\x0bu}qF(h\x02K\x19h\x03K2h\x04G@H\r:\x06\xd
3\xa0nh\x05G@Q=:\x06\xd3\xa0mh\x06G?\xf6\xef\xa9nIz4h\x07h\ru}qG(h\x02K
\x19h\x03KKh\x04G@L\xa1\xb4\xe8\x1bN\x83h\x05G@MM\xa7@\xdat\x0fh\x06G?
\xf0`\x16i\x98O\xfeh\x07h\x08u}qH(h\x02K\x19h\x03KKh\x04G@L333333h\x05G
@P%\x1e\xb8Q\xeb\x85h\x06G?\xf2R\x10\xb4\x1b\x98\xe0h\x07h\x0bu}qI(h\x0
2K\x19h\x03KKh\x04G@K\xf5UUUUUh\x05G@P\xb9\x99\x99\x99\x99\x9ah\x06G?\x
f3$\x8c\xb4a\xf9qh\x07h\ru}qJ(h\x02K2h\x03K\x19h\x04G@J\xdbN\x81\xb4\xe
8\x1ch\x05G@M#i\xd06\x9d\x02h\x06G?\xf1[\xfc\x0f`\xe0\xb3h\x07h\x08u}qK
(h\x02K2h\x03K\x19h\x04G@I\x9dp\xa3\xd7\n=h\x05G@N\xfe\xb8Q\xeb\x85\x1f
h\x06G?\xf3\\K\xc1o\xc8\xd6h\x07h\x0bu}qL(h\x02K2h\x03K\x19h\x04G@KK\x1
7\xe4\xb1~Lh\x05G@Q8Q\xeb\x85\x1e\xb9h\x06G?\xf40\x86\x10\xc2\x18Ch\x07
h\ru}qM(h\x02K2h\x03K2h\x04G@Pib\xfc\x96/\xcah\x05G@Qu\x8b\xf2X\xbf%h\x
06G?\xf1\x05n\xfc\x05\xb81h\x07h\x08u}qN(h\x02K2h\x03K2h\x04G@N\xe4DDDD
Dh\x05G@R\xd7\n=p\xa3\xd7h\x06G?\xf3\x84\x14\xd3\xfbj\x1eh\x07h\x0bu}qO
(h\x02K2h\x03K2h\x04G@Q),_\x92\xc5\xf9h\x05G@T\xc9b\xfc\x96/\xc9h\x06G?
\xf3al\xc9\x89\xbaBh\x07h\ru}qP(h\x02K2h\x03KKh\x04G@Q}\x03i\xd06\x9eh

\x05G@QP\xdat\r\xa7Ah\x06G?\xef\xaf2H\\\xb5\x9bh\x07h\x08u}qQ(h\x02K2h
\x03KKh\x04G@R\x1c\xcc\xcc\xcc\xcc\xcdh\x05G@SG@\xdat\r\xa8h\x06G?\xf1
\x07\xa4\xe20t\xa7h\x07h\x0bu}qR(h\x02K2h\x03KKh\x04G@Sq~K\x17\xe4\xb2h
\x05G@T\xdb\x85\x1e\xb8Q\xebh\x06G?\xf1)\xe9\xca\xe5/-h\x07h\ru}qS(h\x0
2KKh\x03K\x19h\x04G@K\xc3i\xd06\x9d\x03h\x05G@L\xcf\\(\xf5\xc2\x90h\x06
G?\xf0\x9aj\xe6U\xe0Xh\x07h\x08u}qT(h\x02KKh\x03K\x19h\x04G@K\xac_\x92
\xc5\xf9,h\x05G@P\xa7\xae\x14z\xe1Hh\x06G?\xf3BL\x1aeC\xe7h\x07h\x0bu}q
U(h\x02KKh\x03K\x19h\x04G@P\xd06\x9d\x03i\xd0h\x05G@P\xc7wwwwxh\x06G?\x
ef\xefZWK\xb9[h\x07h\ru}qV(h\x02KKh\x03K2h\x04G@P\xd3i\xd06\x9d\x04h\x0
5G@R%\xf9,_\x92\xc7h\x06G?\xf1A\xf1kS\xb8\th\x07h\x08u}qW(h\x02KKh\x03K
2h\x04G@R&fffffh\x05G@S\xac\x96/\xc9b\xfdh\x06G?\xf1W\xf7Y`m\xd5h\x07h
\x0bu}qX(h\x02KKh\x03K2h\x04G@R\xb0\xdat\r\xa7Ah\x05G@T\xf6ffffgh\x06G?
\xf1\xf1\xd2\xf6\x0e\xffkh\x07h\ru}qY(h\x02KKh\x03KKh\x04G@Qq~K\x17\xe4
\xb2h\x05G@REUUUUUh\x06G?\xf0\xc2O\xbdw\xe7Wh\x07h\x08u}qZ(h\x02KKh\x03
KKh\x04G@T.K\x17\xe4\xb1~h\x05G@T-\xdd\xdd\xdd\xdd\xdeh\x06G?\xef\xffR
\xcd\xa6`\xdbh\x07h\x0bu}q[(h\x02KKh\x03KKh\x04G@V\xb3\xa0m:\x06\xd4h\x
05G@W\x90\xdat\r\xa7Ah\x06G?\xf0\x9b\xebSwf\xc9h\x07h\ru}q\\(h\x02K\x19
h\x03K\x19h\x04G@GV/\xc9b\xfc\x95h\x05G@G\xe8\xf5\xc2\x8f\\(h\x06G?\xf0
d\xa1T\x81&\xdfh\x07h\x08u}q](h\x02K\x19h\x03K\x19h\x04G@A\x14\xe8\x1bN
\x81\xb5h\x05G@H\x0e\xee\xee\xee\xee\xeeh\x06G?\xf6\x88\xed\xfd\xa6\xed
\xa3h\x07h\x0bu}q^(h\x02K\x19h\x03K\x19h\x04G@A\xe2"""!h\x05G@H\x11\x1
1\x11\x11\x11\x10h\x06G?\xf5\x88;+/\x00=h\x07h\ru}q_(h\x02K\x19h\x03K2h
\x04G@I\xcd:\x06\xd3\xa0nh\x05G@M\x8f\xc9b\xfc\x96/h\x06G?\xf2T\xe5\xdb
\x1f\xb3Dh\x07h\x08u}q`(h\x02K\x19h\x03K2h\x04G@E\xb6\x9d\x03i\xd06h\x0
5G@M\xaa=p\xa3\xd7\nh\x06G?\xf5\xdc\x00P|\x80\x97h\x07h\x0bu}qa(h\x02K
\x19h\x03K2h\x04G@H4\r\xa7@\xdath\x05G@Q\x07\xe4\xb1~K\x18h\x06G?\xf6\x
84Z\xae=F\xc3h\x07h\ru}qb(h\x02K\x19h\x03KKh\x04G@L8Q\xeb\x85\x1e\xb9h
\x05G@MrX\xbf%\x8b\xf2h\x06G?\xf0\xb2\x0b}\x9ct[h\x07h\x08u}qc(h\x02K\x
19h\x03KKh\x04G@LK\x17\xe4\xb1~Kh\x05G@P"""""h\x06G?\xf2?7@\xe7\xa1kh
\x07h\x0bu}qd(h\x02K\x19h\x03KKh\x04G@N\xad\xa7@\xdat\x0eh\x05G@Q7\xe4
\xb1~K\x18h\x06G?\xf1\xf5\xca\xad|<\xeah\x07h\ru}qe(h\x02K2h\x03K\x19h
\x04G@K\xfbN\x81\xb4\xe8\x1bh\x05G@M\xfa\x06\xd3\xa0m<h\x06G?\xf1$\x07
\xfdp5\xech\x07h\x08u}qf(h\x02K2h\x03K\x19h\x04G@E\xdc\x96/\xc9b\xfch\x
05G@M\x03\xd7\n=p\xa3h\x06G?\xf5<?w\xda\x0fCh\x07h\x0bu}qg(h\x02K2h\x03
K\x19h\x04G@H\xf6\x9d\x03i\xd07h\x05G@P\xe7\xe4\xb1~K\x18h\x06G?\xf5\xa
b\xd7\x1d\xd7\x98[h\x07h\ru}qh(h\x02K2h\x03K2h\x04G@P0\xa3\xd7\n=qh\x05
G@Q\x15UUUUUh\x06G?\xf0\xe2\x02l\xae\xc7 h\x07h\x08u}qi(h\x02K2h\x03K2h
\x04G@N\x04DDDDDh\x05G@R\xeeK\x17\xe4\xb1~h\x06G?\xf4.\x82_\xb0\xda\xc9
h\x07h\x0bu}qj(h\x02K2h\x03K2h\x04G@On\x81\xb4\xe8\x1bOh\x05G@T\xe6/\xc
9b\xfc\x96h\x06G?\xf5F\xed%\'7lh\x07h\ru}qk(h\x02K2h\x03KKh\x04G@QU\x1e
\xb8Q\xeb\x85h\x05G@Q\xf5\xc2\x8f\\(\xf6h\x06G?\xf0\x94JJ\xf5\x1bph\x07
h\x08u}ql(h\x02K2h\x03KKh\x04G@Qsi\xd06\x9d\x03h\x05G@Sd\xb1~K\x17\xe4h
\x06G?\xf1\xc7\xef\xdc\xfc\xf4\xbdh\x07h\x0bu}qm(h\x02K2h\x03KKh\x04G@S
d\xb1~K\x17\xe4h\x05G@T\xfa\xe1G\xae\x14{h\x06G?\xf1O\x1dN\x0e\xf5\xe2h
\x07h\ru}qn(h\x02KKh\x03K\x19h\x04G@LQ\x11\x11\x11\x11\x10h\x05G@M\xb4z
\xe1G\xae\x15h\x06G?\xf0\xc8\xd2w\'\xbf\xa2h\x07h\x08u}qo(h\x02KKh\x03K
\x19h\x04G@P\x08\xf5\xc2\x8f\\)h\x05G@P\xe7\xae\x14z\xe1Hh\x06G?\xf0\xd
e;\xdey,\xc8h\x07h\x0bu}qp(h\x02KKh\x03K\x19h\x04G@O\x07@\xdat\r\xa8h\x
05G@Q\x19,_\x92\xc5\xf9h\x06G?\xf1\xa2?\x1d\xd3]Zh\x07h\ru}qq(h\x02KKh
\x03K2h\x04G@Q\x8d\x03i\xd06\x9ch\x05G@Q8\x88\x88\x88\x88\x89h\x06G?\xe
fe\xf8>\xb8}\xbeh\x07h\x08u}qr(h\x02KKh\x03K2h\x04G@R?\xc9b\xfc\x960h\x
05G@S~K\x17\xe4\xb1\x7fh\x06G?\xf1\x17@`\x83Mmh\x07h\x0bu}qs(h\x02KKh\x
03K2h\x04G@S\xceK\x17\xe4\xb1\x7fh\x05G@U\x05\xc2\x8f\\(\xf6h\x06G?\xf0
\xfb\x9d\xb9%\x94\xb2h\x07h\ru}qt(h\x02KKh\x03KKh\x04G@Rtz\xe1G\xae\x15
h\x05G@RqG\xae\x14z\xe0h\x06G?\xef\xfas\x8c\xdd\xaa\xf4h\x07h\x08u}qu(h
\x02KKh\x03KKh\x04G@U\x0c\xcc\xcc\xcc\xcc\xcch\x05G@U\x84\xb1~K\x17\xe4
h\x06G?\xf0[!\\\xf4]\x90h\x07h\x0bu}qv(h\x02KKh\x03KKh\x04G@Vp\xa3\xd7

\n=qh\x05G@W\xca\x06\xd3\xa0m:h\x06G?\xf0\xf6C\xd7<\x15Kh\x07h\ru}qw(h
\x02K\x19h\x03K\x19h\x04G@D\xd3\xa0m:\x06\xd5h\x05G@Fp\xa3\xd7\n=ph\x06
G?\xf1=K\xb7\xe3\'\xa8h\x07h\x08u}qx(h\x02K\x19h\x03K\x19h\x04G@B\xd4z
\xe1G\xae\x15h\x05G@G\xfeK\x17\xe4\xb1\x7fh\x06G?\xf4c\'\xbea@}h\x07h\x
0bu}qy(h\x02K\x19h\x03K\x19h\x04G@C%\x8b\xf2X\xbf%h\x05G@H*=p\xa3\xd7\n
h\x06G?\xf41\x8fT\xdb\xa1\xe0h\x07h\ru}qz(h\x02K\x19h\x03K2h\x04G@I\xf5
\xc2\x8f\\(\xf7h\x05G@L\xf2X\xbf%\x8b\xf2h\x06G?\xf1\xd7=g7\x17\x88h\x0
7h\x08u}q{(h\x02K\x19h\x03K2h\x04G@E+\x17\xe4\xb1~Jh\x05G@M\xf2X\xbf%\x
8b\xf3h\x06G?\xf6\xa2\x93\xd2\xc5*<h\x07h\x0bu}q|(h\x02K\x19h\x03K2h\x0
4G@K\x05\x1e\xb8Q\xeb\x84h\x05G@Qg@\xdat\r\xa7h\x06G?\xf4\x9cdM\x90t9h
\x07h\ru}q}(h\x02K\x19h\x03KKh\x04G@M=\xdd\xdd\xdd\xdd\xdeh\x05G@N\x11
\xeb\x85\x1e\xb8Rh\x06G?\xf0t\x07-\xc5\xaa<h\x07h\x08u}q~(h\x02K\x19h\x
03KKh\x04G@N\r\xa7@\xdat\x0eh\x05G@P\x9b\xbb\xbb\xbb\xbb\xbbh\x06G?\xf1
\xaf"L\xc6\xfc"h\x07h\x0bu}q\x7f(h\x02K\x19h\x03KKh\x04G@Jwwwwwwh\x05G@
Q,\x96/\xc9b\xfch\x06G?\xf4\xc3\xc5\xb4\xf0\xe9,h\x07h\ru}q\x80(h\x02K2
h\x03K\x19h\x04G@K\xeb\x85\x1e\xb8Q\xedh\x05G@N\x04DDDDDh\x06G?\xf13\x9
7Y\xbc\x06\xd8h\x07h\x08u}q\x81(h\x02K2h\x03K\x19h\x04G@G\x02"""""h\x05
G@M\x18\xbf%\x8b\xf2Xh\x06G?\xf4;\xdc,HxPh\x07h\x0bu}q\x82(h\x02K2h\x03
K\x19h\x04G@G\xe2\xfc\x96/\xc9ch\x05G@P\xcb\xbb\xbb\xbb\xbb\xbch\x06G?
\xf6\x80-\xba\x1f\x94\x8bh\x07h\ru}q\x83(h\x02K2h\x03K2h\x04G@O\xd4z\xe
1G\xae\x15h\x05G@Q\xe8\xbf%\x8b\xf2Yh\x06G?\xf2\x01;\xb4/\x13\xa7h\x07h
\x08u}q\x84(h\x02K2h\x03K2h\x04G@O"\xfc\x96/\xc9bh\x05G@RD\xb1~K\x17\xe
6h\x06G?\xf2\xc6]td\xbe\xa5h\x07h\x0bu}q\x85(h\x02K2h\x03K2h\x04G@Nm\xa
7@\xdat\x0eh\x05G@U"\x8f\\(\xf5\xc1h\x06G?\xf6:\x05\xea\xd3\xa5\xf9h\x0
7h\ru}q\x86(h\x02K2h\x03KKh\x04G@P\xeat\r\xa7@\xdbh\x05G@Q\xe2\xfc\x96/
\xc9ch\x06G?\xf0\xeb\x13\xdc\xf0\xadTh\x07h\x08u}q\x87(h\x02K2h\x03KKh
\x04G@Q\xd7\xe4\xb1~K\x17h\x05G@S\xe5\xf9,_\x92\xc6h\x06G?\xf1\xd7\xbb
\x84;~\x1dh\x07h\x0bu}q\x88(h\x02K2h\x03KKh\x04G@S\xa9b\xfc\x96/\xcah\x
05G@VX\xf5\xc2\x8f\\)h\x06G?\xf2/\x86!`\x11\\h\x07h\ru}q\x89(h\x02KKh\x
03K\x19h\x04G@N\x00m:\x06\xd3\xa1h\x05G@N6/\xc9b\xfc\x97h\x06G?\xf0\x1c
\xab\x9f\xceeIh\x07h\x08u}q\x8a(h\x02KKh\x03K\x19h\x04G@M\xd4\r\xa7@\xd
auh\x05G@OM:\x06\xd3\xa0mh\x06G?\xf0\xcaQ\x1a\xa6\x03Xh\x07h\x0bu}q\x8b
(h\x02KKh\x03K\x19h\x04G@N\x94\r\xa7@\xdauh\x05G@O\xb2X\xbf%\x8b\xf2h\x
06G?\xf0\x95\xcdK?\xf2\x9ah\x07h\ru}q\x8c(h\x02KKh\x03K2h\x04G@Q\x91\x1
1\x11\x11\x11\x11h\x05G@Q\xf5UUUUVh\x06G?\xf0[S\x11\x00|\\h\x07h\x08u}q
\x8d(h\x02KKh\x03K2h\x04G@S-\xa7@\xdat\x0eh\x05G@TG\xe4\xb1~K\x19h\x06
G?\xf0\xebw92\xa8=h\x07h\x0bu}q\x8e(h\x02KKh\x03K2h\x04G@S\xd8Q\xeb\x85
\x1e\xb9h\x05G@T\x82\xfc\x96/\xc9dh\x06G?\xf0\x89\x99\x87\xfc\xbbeh\x07
h\ru}q\x8f(h\x02KKh\x03KKh\x04G@Q\xbd\xa7@\xdat\rh\x05G@R\x8f\\(\xf5\xc
2\x90h\x06G?\xf0\xbd!\x08[|th\x07h\x08u}q\x90(h\x02KKh\x03KKh\x04G@T\x8
7\xae\x14z\xe1Hh\x05G@U\x1a\x06\xd3\xa0m:h\x06G?\xf0r\x0e\x0c\x8d|\xe7h
\x07h\x0bu}q\x91(h\x02KKh\x03KKh\x04G@W\x9e\xee\xee\xee\xee\xefh\x05G@X
\x8d:\x06\xd3\xa0mh\x06G?\xf0\xa1i\x8dQ*\xb1h\x07h\rue.'

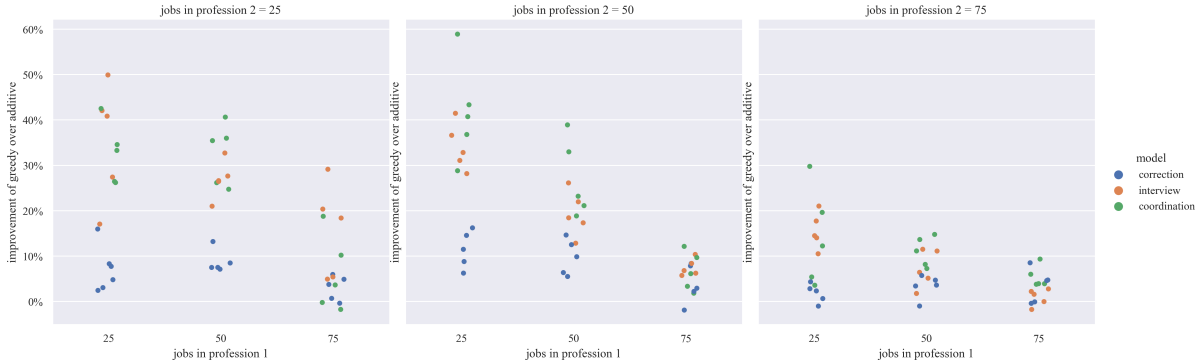
In [45]:
```
import numpy as np
matchings = np.array(data)
np.savez("matchingslist", data)
```

In [285]:
```
plot()
```



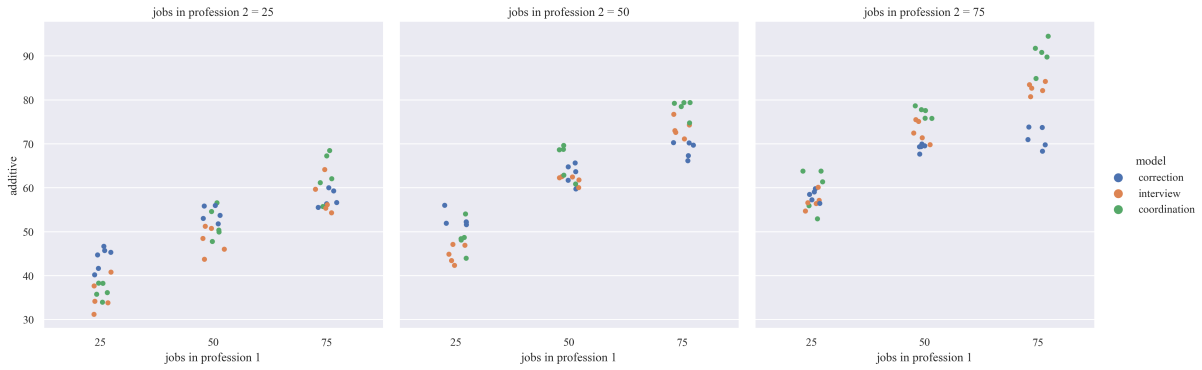
In [51]:
```
def _format_y(ratio):
    return f"{ratio-1:,.0%}"

def plot():
    d = pd.DataFrame(data)
    g = seaborn.catplot(x="jobs in profession 1",
                        y="greedy / additive", hue="model",
                        col="jobs in profession 2", data=d)
    for ax in g.axes[0]:
        vals = ax.get_yticks()
        ax.set_yticklabels([_format_y(x) for x in vals])
        ax.set_ylabel("improvement of greedy over additive")
    g.savefig("job_availabilityfinal.pdf")
```

In [52]:
```
plot()
```



In [53]:
```
def plot_absolute_utilities():
    d = pd.DataFrame(data)
    g = seaborn.catplot(x="jobs in profession 1", y="additive",
                        hue="model", col="jobs in profession 2",
                        data=d)
    g.savefig("job_availability_additivefinal.pdf")
```

In [54]:
```
plot_absolute_utilities()
```



In [ ]: