10-9-2023

Match Stability with a Costly and Flexible Number of Positions

James Gilmore

David Porter

Match Stability with a Costly and Flexible Number of Positions

Comments

ESI Working Paper 23-10

Match Stability with a Costly and Flexible Number of Positions

James Gilmore
David Porter

October 9, 2023

Abtract: One of the primary objectives of two-sided matching systems is to facilitate the pairing of two groups of agents in a manner that eliminates any incentive for pair deviation. Such challenges are quite prevalent and can have significant and long-lasting ramifications for participants, including students applying to colleges. While much of the existing research in this field addresses the problem using fixed quotas, real-world applications, like college admissions, demonstrate that this is not always applicable. We introduce the concept of slot stability, recognizing the potential motivation for organizations to modify their quotas after the matching process. We propose two algorithms designed to create stable and slot stable matches by employing flexible, endogenous quotas to address this issue. Additionally, we demonstrate that our algorithm aligns with the concerns raised by colleges implementing waitlist systems, effectively mitigating behaviors that can lead to unstable outcomes.

1 Introduction

Economic research has played a pivotal role in shaping the development of market institutions, with a particular emphasis on matching institutions. The conventional framework for addressing the matching problem involves entities offering a set number of positions or quotas that need to be filled by applicants. Within this framework, each applicant can be assigned to at most one position. Matching institutions have been meticulously crafted to tackle these intricate challenges, aiming to provide a stable solution.

The concept of match stability was originally introduced by Gale and Shapley [1] in their groundbreaking paper. They define a match as stable when there is no compelling incentive for a pair of participants to switch their assignments. Additionally, Gale and Shapley introduced the deferred acceptance algorithm [1], a widely employed method for achieving stable matches between applicants and organizations. In this algorithm, applicants submit rank order lists (ROLs) of organizations based on their preferences, while organizations similarly rank applicants according to their preferences. The outcome of this algorithm results in a stable match when ROLs truthfully reveal the rankings of payoffs.

The basic matching problem has evolved and extended to cases where preferences are more complex. For example, the student-project allocation problem, described by Abraham et al. [2], deals with matching students to projects that can have overlapping lecturers, while taking into account individual preferences and class capacity constraints. In this environment, the matching algorithm was modified to ensure stability. Another example is in the National Resident Matching Program, which assigns interns to different hospitals and specialties. At first, the algorithm treated every individual's preference as independent of any other individual's preference and gave a stable matching in that environment. However, couples in the match may have joint preferences because they want to be near each other. The deferred acceptance algorithm does not consider this and can produce unstable outcomes. In this environment, Roth and Peranson [3] proposed a new matching algorithm that incorporates couple preferences, although it does not guarantee a stable match. Nonetheless, computational experiments demonstrate that the algorithm's outcomes closely approximate stability.

The matching literature currently defines stability under the assumption of fixed quotas. Organizations state the maximum number of applicants they are willing to accept before the matching process begins. Yet, it remains unclear how these quotas are determined and how organizations value applicants relative to their quotas. Rios et al. [4] examined the Chilean college admission system, where the maximum number of slots can exceed the preset quota. Matches are based entirely on academic scores which can have ties. Therefore, quotas can be exceeded if there is a tie between the accepted worst candidate and any other candidate who wants to join, in which case they must accept all such candidates. However, the process starts by posting a quota and then adjusting it in light of scores. Limaye and Nasre [5] explore cases where all applicants must be accepted with costly slots. They then minimize the total cost to get a stable match with minimal cost. However, this does not address the incentive for the organization to accept these quotas. Here, there is minimal cost, yet there may be some excellent candidates the organization would be willing to accept at a higher cost.

In the context of university admissions, educational institutions often grapple with a challenging dilemma. They frequently find themselves with a surplus of highly qualified applicants, compelling them to consider increasing the number of admitted students beyond their initial enrollment quotas. However, this decision is not taken lightly, as universities must balance the advantages of admitting exceptional students and the practical constraints of managing undergraduate enrollment while considering campus resource costs. To navigate this complex scenario, universities have implemented waiting lists for students who have not yet received acceptance offers.

A similar dilemma arises when universities are in the process of recruiting new faculty members. In this case, while the administration may provide a specific number of available positions, academic departments may argue for additional positions if confronted with a pool of high-quality candidates. The ability to assess both the quality of applicants and the associated costs of creating additional positions becomes pivotal in making these matching decisions.

In this paper, we show that if we expand stability to include organizations offering a different number of positions, the current algorithms are not necessarily stable. We show how a small change to the deferred acceptance algorithm allows for endogenous numbers of slots by organizations while guaranteeing this expanded stability. In particular, we propose a matching mechanism that allows ROLs to accommodate these trade-offs and ensure a stable match that is also slot stable. By slot stable, we mean that every organization has no incentive to deviate in their number of openings. We also show that our matching mechanism takes into account the concerns of organizations in a wait list system and provides a solution to the endogenous quota problem.

2 The Environment

Applicants are denoted as a, with indices $i = 1, 2, ..., n$, and organizations are denoted as o, with indices $j = 1, 2, ..., m$. Each organization o_j has a number of positions or *slots* to fill s_j. Each applicant can fill one slot with at most one organization, and the set of these applicants admitted to o_j is denoted as A_j. Let $V_i(o_j)$ denote applicant a_i's value if they are matched with o_j. Let $Z_j(a_i)$ denote o_j's value if they are matched with a_i. Both V_i and Z_j are one-to-one functions. Every a_i and o_j is individually rational and defined by refusing all matches such that $V_i(\emptyset) > V_i(o_j)$ or $Z_j(\emptyset) > Z_j(a_i)$, i.e., applicants only rank organizations that improve their value over remaining unmatched.

Organization o_j has a non-decreasing convex total cost $C_j(s_j)$ of filling slots. Specifically $C_j(s_j+1) \geq C_j(s_j)$ and $C_j(s_j+2) - C_j(s_j+1) \geq C_j(s_j+1) - C_j(s_j)$. Denote $MC_j(s_j)$ to be the marginal cost of filling slot number s_j defined by $C_j(s_j) - C_j(s_j-1)$. We also assume every a_i ranks the organizations based only on V_i where a_i prefers o_j over o_k if and only if $V_i(o_j) > V_i(o_k)$. Likewise, o_j ranks the applicants based only on Z_j where o_j prefers a_i over a_k if and only if $Z_j(a_i) > Z_j(a_k)$.

Lastly, we define $\delta_j(a_i) = 1$ if o_j accepts a_i in a match and 0 otherwise. The current deferred acceptance algorithm does not guarantee stability in this environment. Below is an example illustrating the issue with the fixed quota assumption.

Suppose we have two organizations o_1, o_2, and three applicants a_1, a_2, a_3. Both organizations have the same values Z_j and costs C_j with $Z_j(a_1) = 5$, $Z_j(a_2) = 4$,$Z_j(a_3) = 3$, and $MC_j(1) = 2$, $MC_j(2) = 3.5$. For the applicants their preferences are defined by $V_1(o_2) > V_1(o_1)$, $V_2(o_2) > V_2(o_1)$, $V_3(o_1) > V_3(o_2)$. The tables have the participants listed in the columns, while the rows depict the cost, values, or ranking of the object listed in the row.

	o_1	o_2
a_1	5	5
a_2	4	4
a_3	3	3

Table 1: Organization $Z_j(a_i)$

	o_1	o_2
Slot 1	2	2
Slot 2	3.5	3.5

Table 2: Organization $MC_j(x)$

o_1	o_2
a_1	a_1
a_2	a_2
a_3	a_3

Table 3: Organization ROLs

a_1	a_2	a_3
o_2	o_2	o_1
o_1	o_1	o_2

Table 4: Applicant ROLs

The applicant-proposing deferred acceptance algorithm gives the following output.

o_1	o_2
a_3	a_1
$\emptyset$	a_2

Table 5: First Match; $s_j = 2$

This yields a stable match, and neither organization has any incentive to want to change its quota, s_j. However, if a_2's preference was $V_2(o_1) > V_2(o_2)$, their ROL would now be o_1, o_2, and the applicant proposing deferred acceptance match would be the one found in Table 6. Notice that with o_1 having two slots filled, the value of a_3 in slot 2 has a value of 3 but a marginal cost of 3.5, resulting in a loss of .5. Because of this, o_1 would prefer to leave the second slot unfilled since $MC_1(s_2) > Z_1(a_3)$. Here o_1 set their quota too high.

o_1	o_2
a_2	a_1
a_3	$\emptyset$

Table 6: Alternate match: o_1 takes a loss; $s_j = 2$

Now, suppose organizations have the same costs and values as before, but the quotas are 1 for each organization. Applicant preferences are the same as the first example in table 4: $V_1(o_2) > V_1(o_1)$, $V_2(o_2) > V_2(o_1)$, $V_3(o_1) > V_3(o_2)$. Running the deferred acceptance algorithm would give the following match.

o_1	o_2
a_2	a_1

Table 7: o_2 misses a positive payoff; $s_j = 1$

This match is stable; however, o_2 can do better. Here o_2 would be willing to open a slot for a_2 and a_2 prefers o_2 over their current match, which would cause both to be better off. Here, o_2 set their quota too low.

These examples demonstrate that another form of stability concerning organization quotas should be addressed. First, if organization o_j stands to gain by adding a slot for an a_i matched with some o_u that would prefer to be matched with o_j, it is slot unstable. Second, if organization o_j profits by eliminating a slot and terminating an a_i in A_j, it is slot unstable. Hence, we offer the following definition.

Definition: A match is said to be *slot stable* if and only if

$$(1)\quad Z_j(A_j) - C_j(s_j) \geq Z_j(A_j \cup a_i) - C_j(s_j + 1) \quad \forall a_i \notin A_j, a_i \in A_u, \quad V_i(o_u) < V_i(o_j), \quad \forall j \in 1, 2, ..., m$$

and

$$(2)\quad Z_j(A_j) - C_j(s_j) \geq Z_j(A_j \setminus a_i) - C_j(s_j - 1) \quad \forall a_i \in A_j, \quad \forall j \in 1, 2, ..., m$$

This can also be written in terms of marginal costs.[1]

$$(1a)\quad MC_j(s_j + 1) \geq Z_j(a_i) \quad \forall a_i \notin A_j, V_i(o_j) > V_i(o_u)$$

and

$$(2a)\quad Z_j(a_i) \geq MC_j(s_j) \quad \forall a_i \in A_j$$

3 Matching Mechanisms

This section assumes that applicants and organizations submit ROLs consistent with their payoffs.[2]

3.1 Endogenous Number of Positions Applicant-Proposing Algorithm (ENPAP)[3]

3.1.1 Inputs

Applicants submit ROLs listing organizations from their most to least preferred that are better than not being matched at all. For the organizations, we will need an adjusted ranked order list where organizations provide a *cutoff list* of rankings. First, o_j lists all their top candidates X_j, which they would be willing to accept, given costs, if they were all matched. The cutoff is the cardinality of X_j defined here as n_j. This n_j is the upper bound of n where $B_j(n)$ is the set of applicants that satisfy the cutoff rank of n such that $Z_j(a_i) > MC_j(n)$. Then, o_j lists the set of applicants $B_j(n)$ from $n = 1$ to $n = n_j$ they would accept if matched with $n-1$ other higher ranking candidates. In other words, it is the set of applicants that o_j would accept if they had to take on the marginal cost at slot n. By doing this for cutoffs n_j to 1, the mechanism can create o_j's ROL. This method creates the ROL such that if a_i is in $B_j(n)$, they rank above n in the list. Otherwise, they must rank below.

For example, using the valuations from our first example, each organization has the following costs and applicant values: $Z_j(a_1) = 5$, $Z_j(a_2) = 4$, $Z_j(a_3) = 3$, $MC_j(1) = 2$, $MC_j(2) = 3.5$, $MC_j(3) = 7$. Creating

[1] Justification is shown in Appendix A.

[2] Just like with G-S matching, the non-proposing side may not be incentivized to reveal their true rankings. Our mechanisms ensure that the match with truthful rankings will be stable.

[3] The stability results for the organization proposing case can be found in Appendix B.

the best possible list for o_1 and o_2 results in $n_1 = n_2 = 2$. This is because the best outcome for both is to be matched with a_1 and a_2. Here both o_1 and o_2 would take both a_1 and a_2 if they had to pay the marginal cost in slot 2 to match with them. So far we have [a_1, a_2, 2, ..., 1]. Next, we check for slot n-1, which in this case is 1. Both organizations would accept all three candidates if they only had to pay the marginal cost of slot 1. Therefore, the ROL for o_1 and o_2 would be written as [a_1, a_2, 2, a_3, 1].

3.1.2 Algorithm[4]

Using the notation from the G-S algorithm, all applicants propose to the organization at the top of their ROL. Then every o_j looks at their lowest value applicant a_k that proposed to them and checks if a_k is acceptable in slot s_j by looking at o_j's ROL. If a_k ranks lower than s_j, o_j rejects a_k and o_j is removed from a_k's ROL. All applicants are tentatively accepted if a_k ranks higher than s_j. If there is an a_k such that a_k is unmatched and has any o_k remaining in their ROL, they propose to their top remaining organization, and so forth. To illustrate this, we use the applicant valuations $V_1(o_1) > V_1(o_2)$, $V_2(o_1) > V_2(o_2)$, $V_3(o_1) > V_3(o_2)$ and the ROL [a_1, a_2, 2, a_3, 1] for both o_1 and o_2. First, each applicant proposes to their highest valued, individually rational organization depicted in Table 8.

o_1	o_2
a_1	$\emptyset$
a_2	$\emptyset$
a_3	$\emptyset$

Table 8: Applicants first proposal

Looking at the ROL of o_1, [a_1, a_2,2,a_3,1], we eliminate the lowest ranking applicant a_3, and a_1 and a_2 are tentatively accepted. After being rejected from o_1, a_3 proposes to o_2 who accepts them since a_3 was ranked if there is only one slot to fill for o_2. This yields the final match in Table 9.

o_1	o_2
a_1	a_3
a_2	$\emptyset$

Table 9: Final Applicant Proposing Match

Theorem 1.1: ENPAP results in a stable match

Proof: Suppose the ENPAP match is unstable, then $\exists a_i, o_j$ matched with o_u, a_u such that $V_i(o_j) > V_i(o_u)$ and $Z_j(a_i) > Z_j(a_u)$. For a_i and o_j to not be matched with each other, either a_i never proposed to o_j or o_j rejected a_i.

If a_i never proposed to o_j, one of two scenarios could have happened.

(ia) a_i never put o_j on their list. If o_j is not on a_i's list, then $V_i(\emptyset) > V_i(o_j)$. All o_k ranked by a_i must satisfy $V_i(\emptyset) < V_i(o_k)$. Therefore regardless of a_i being matched with no one or any o_k in their ROL, $V_i(o_j) > V_i(o_u)$ is false.

(ib) a_i never proposed o_j on their ROL. For this to happen, since a_i applies to their highest ranked organization to their lowest ranked organization, a_i must have stopped when matched with o_u ranked higher than o_j such that $V_i(o_u) > V_i(o_j)$.

[4]Python code of this algorithm can be found in Appendix C.

(ii) o_j rejected a_i. If $Z_j(a_i) < Z_j(\emptyset)$, then the algorithm cannot make a match where $Z_j(a_i) > Z_j(a_u)$. Since the algorithm only rejects the lowest ranked applicants, all other applicants tentatively accepted in the organization at the time must have ranked higher than a_i and $MC_j(s) > Z_j(a_i)$ where s in the number of tentatively accepted applicants. For o_j to still want a_i compared to one of the applicants they were matched with, someone ranked even lower than a_i must have been accepted later. If a_u ranks first to s among A_j, it follows that a_u must be ranked above at least one other a_k that was tentatively accepted while a_i was rejected meaning $Z_j(a_u) > Z_j(a_k) > Z_j(a_i)$. If a_u was tentatively accepted with s or higher slots, then $Z_j(a_u) > MC_j(s) > Z_j(a_i)$. This would mean that in either case, there does not exist a blocking pair as $Z_j(a_i) > Z_j(a_u)$ is false. Q.E.D.

Theorem 1.2: ENPAP results in a slot stable match

Proof: Assume ENPAP results in slot instability, then by definition $\exists a_k, o_j$ such that
(1) $Z_j(A_j) - C_j(s_j) < Z_j(A_j \cup a_i) - C_j(s_j + 1)$ and $V_i(o_u) < V_i(o_j)$,

or

(2) $Z_j(A_j) - C_j(s_j) < Z_j(A_j \setminus a_i) - C_j(s_j - 1)$.

(1) If the first inequality is true, then $\exists a_i$ such that $MC(s_j + 1) < Z_j(a_i)$ that ranks worse than all the other tentatively accepted applicants or $\exists a_i, a_u$ such that $Z_j(a_i) > Z_j(a_u)$ and $MC_j(s_j + 1) < Z_j(a_u)$. For the first case, if $V_i(o_u) < V_i(o_j)$, then a_i would have already been matched with o_j as a_i would have proposed to o_j before o_u and not be rejected. For the second case, if $V_i(o_u) < V_i(o_j)$ the match would have been unstable, which is not possible from Theorem 1.1.

(2) If the second inequality is true, $\exists a_k$, that is the lowest value $a_i \in A_j$ matched together such that $MC_j(s_j) > Z_i(a_k)$. However, the ENPAP algorithm rejects all a_i that do not satisfy $MC_j(s_j) < Z_j(a_i)$. Since a_k was not rejected by the algorithm, then $MC_j(s_j) < Z_j(a_k)$ must be true.

Since the algorithm cannot produce a match that satisfies either condition, the ENPAP must give a slot-stable match. Q.E.D.

Among the set of stable and slot stable matches, an *applicant optimal match* is the one that assigns applicants to their highest ranking feasible organization.

Theorem 1.3: ENPAP results in an applicant optimal match

Proof: Using induction and Theorem 1.1, assume that the algorithm does not give an applicant optimal match. That would mean that there exists an applicant a_i that could match with a better organization that did not. Since this is applicant proposing, assume that no applicant has yet been rejected by an organization that is achievable for them. This means that no o_j has rejected any a_i where there exists a stable, slot stable match with a_i matched to o_j. If a_i was rejected for being unacceptable, it is unachievable. If a_i was rejected in favor of a_k, then it is known that the applicant a_k prefers the organization o_u except for those that already rejected them. By the inductive assumption, those organizations are unachievable to a_k. If we consider a hypothetical matching that matches a_i to the o_u and everyone else to an achievable organization, a_k would prefer the o_u and vice versa, making it an unstable match. Q.E.D.

3.2 Endogenous Number of Positions Organization-Proposing Algorithm (ENPOP) [5]

3.2.1 Inputs

We will be using the same inputs of the ROLs as the ENPAP algorithm described in section 3.1.1.

3.2.2 Organization Optimal List

For any o_j with cost function C_j, value function Z_j and potential applicants A, we define the optimal set X_j for o_j as the set that satisfies argmax $\sum_{i=1}^{n} \delta_j(a_i)\, Z_j(a_i) - C_j(\sum_{i=1}^{n} \delta_j(a_i))$. This set is the top n applicants for o_j in A that satisfy $Z_j(a_i) > MC_j(n)$. Let A be all a_i ranked by o_j that would prefer o_j over their current match. This only restricts any o_j from proposing to a a_i such that $V_i(\emptyset) > V_i(o_j)$ or $V_i(o_u) > V_i(o_j)$ where a_i is tentatively matched to o_u.

3.2.3 Algorithm

Step 1: Each organization submits their optimal organization lists.
Step 2: Each a_i chooses their most preferred o_j among those that put a_i on their optimal list.
Step 3: Repeat the process until no applicant has multiple organizations proposing to them.
To illustrate this we use the valuations from before with applicant values resulting $V_1(o_2) > V_1(o_1)$, $V_2(o_2) > V_2(o_1)$, $V_3(o_1) > V_3(o_2)$ and organization values for both organizations leading to their respective ROL being [a_1, a_2, 2, a_3, 1]. First, each o_j submits their optimal organization list, shown in Table 10.

o_1	o_2
a_1	a_1
a_2	a_2

Table 10: Organization Proposing First List

Since both a_1 and a_2 has been proposed to by both o_1 and o_2, they choose between them. In this case both a_1 and a_2 choose o_2. We then repeat the process where o_2 submits the same list, however o_1 submits a new optimal list [a_3] since their preferred candidates a_1 and a_2 are tentatively in o_2's list. This leads to the final match below.

o_1	o_2
a_3	a_1
$\emptyset$	a_2

Table 11: Organization Proposing Match

3.2.4 Wait list Comparison

Consider the following wait list system. Every applicant applies to all organizations with which they are willing to be matched. Then, every organization accepts their top applicants. Organizations also put unaccepted applicants on a wait list. If accepted applicants decline the offer, the organization then sends acceptances to their wait list. From an organization's perspective, this closely models the current college and graduate school admissions.

The ENPOP algorithm closely resembles the wait list system when viewed from this perspective. Initially, each applicant submits applications to all organizations based on their Rank Order List (ROL). Subsequently, each organization selects their top candidates, taking into consideration the trade-off between marginal costs

[5] Proofs of the stability of this algorithm are shown in Appendix B.

and the applicant's preferences.

Following this, each applicant chooses the organization that provides them with the highest value among those who have accepted them. The process then repeats itself, with each organization once again selecting their preferred candidates, who are likely to accept their offers. In this context, the wait list comprises individuals whom the organization would consider if more preferred applicants declined their offers to match with that organization

The ENPOP algorithm enables organizations to fill vacancies left by applicants who choose another organization, resembling the decision-making considerations seen in institutions such as colleges. However, this parallel behavior does not extend to applicants, as they do not necessarily align with the ENPOP framework. This disconnect becomes particularly evident in the widely seen early acceptance and deadline-related decisions, potentially resulting in unstable outcomes.

Let's examine the scenario of early acceptances. When an applicant, denoted as a_i, accepts an early offer from organization o_j, there are two possible scenarios to consider in their Rank Order List (ROL). First, if there is no organization o_u ranked above o_j in a_i's ROL, it reflects a_i's alignment with the ENPOP framework, as they have secured their best match and have no incentive to deviate.

However, if such an organization o_u exists in a_i's ROL, a potential exists for o_u to extend an offer to a_i. However, if a_i has already accepted o_j's offer, they may find themselves unable to switch to their preferred organization. This situation could lead to an unstable outcome.

Furthermore, we must consider the impact of deadline acceptances. Let's consider two organizations, o_1 and o_2, both of which have sent acceptances to a_1, a_2, and placed a_3 on their respective wait lists. If both a_1 and a_2 delay their decisions until the last possible moment to choose o_1, there may not be enough time for o_2 to send an acceptance offer to a_3 from the wait list, leaving insufficient time for a_3 to make a decision. This dynamic introduces potential instability not observed in either the ENPOP or ENPAP frameworks.

4 Conclusion

We have successfully developed a new matching algorithm that incorporates the cost of supplying slots to be assigned to applicants. This innovative algorithm ensures stable outcomes by incorporating cutoff points in ROLs to account for the cost of supplying slots. Additionally, given the nature of the environment with costly slots, we have defined the requirement for our algorithm to be slot stable. This new concept requires organizations not to be incentivized to change their number of available slots unilaterally. We have also shown that our algorithm is comparable to the current wait list system used in college and graduate school admissions when looking at organization concerns. Yet, it removes the possibility of potentially preemptive behavior, leading to a lower possibility of unstable matches.

5 References

1. D. Gale, L. S. Shapley, College Admissions and the Stability of Marriage. *The American Mathematical Monthly*, 69(1), 9–15 (1962).
2. D. Abraham, R. Irving, D. Manlove, Two algorithms for the Student-Project Allocation problem. *Journal of Discrete Algorithms*, Volume 5, Issue 1, 73-90 (2007).
3. A. Roth, E. Peranson, The Redesign of the Matching Market for American Physicians: Some Engineering Aspects of Economic Design. *American Economic Review*, 89 (4): 748-780 (1999).
4. I. Rios, T. Larroucau, Tomas, G. Parra, R. Cominetti, College Admissions Problem with Ties and Flexible Quotas. *SSRN Electronic Journal* (2014).
5. G. Limaye, M. Nasre, Stable Matchings with Flexible Quotas. *ArXiv.* /abs/2101.04425 (2021)

Appendix A: Marginal Cost Slot Stability Condition

$$(1a)\quad Z_j(A_j) - C_j(s_j) \geq Z_j(A_j \cup a_i) - C_j(s_j + 1)$$

$$C_j(s_j + 1) - C_j(s_j) \geq Z_j(a_i)$$

$$MC_j(s_j + 1) \geq Z_j(a_i)$$

$$(2a)\quad Z_j(A_j) - C_j(s_j) \geq Z_j(A_j \setminus a_i) - C_j(s_j - 1)$$

$$Z_j(a_i) \geq C_j(s_j) - C_j(s_j - 1)$$

$$Z_j(a_i) \geq MC_j(s_j)$$

Appendix B: Endogenous Number of Positions Organization-Proposing Algorithm (ENPOP) Proofs

Theorem 2.1: ENPOP results in a stable match
Proof: Assume that there is a blocking pair a_i and o_j. For this to happen o_j must have put an applicant in their optimal list that is worse than a_i, a_u, in order for $Z_j(a_i) > Z_j(a_u)$ to be satisfied. By optimal list construction, this can only occur if a_i is unavailable. This only happens when $V_i(o_u) > V_i(o_j)$ or $V_i(\emptyset) > V_i(o_j)$ is satisfied. This violates $V_i(o_j) > V_i(o_u)$ therefore ENPOP must result in a stable match. Q.E.D.

Theorem 2.2: ENPOP results in a slot stable match
Proof: For there to be slot instability, there $\exists a_i, o_j$ such that either
(1) $Z_j(A_j) - C_j(s_j) < Z_j(A_j \cup a_i) - C_j(s_j + 1)$ and $V_i(o_u) < V_i(o_j)$, or
(2) $Z_j(A_j) - C_j(s_j) < Z_j(A_j \setminus a_i) - C_j(s_j - 1)$.

(1) If the first inequality is true , then $\exists a_i$ such that $MC(s_j + 1) < Z_j(a_i)$ that ranks worse than all the other tentatively accepted applicants, or an $\exists a_i, a_u$ such that $Z_j(a_i) > Z_j(a_u)$ and $MC_j(s_j + 1) < Z_j(a_u)$. For the first case, if a_i wanted to go to that o_j more than their current match o_u, they would have been already matched as a_i would be qualified to be put on o_j's optimal list and accept the offer. For the second case, if $V_i(o_u) < V_i(o_j)$ the match would have been unstable which violates Theorem 2.2.
(2) If the second inequality is true, $\exists a_i, o_j$ matched together such that $MC_j(s_j) > Z_j(a_k)$ However, for that applicant to have been put into o_j's optimal list with s_j slots, o_j must have ranked before s_j in their ROL meaning $MC_j(s_j) < Z_j(a_k)$.
Since neither inequality can be true, ENPOP must give a slot stable match. Q.E.D

Theorem 2.3 ENPOP results in an organization optimal match *Proof*: Using induction and Theorem 2.1, let's assume that the algorithm does not give an organization optimal match. That would mean that there exists an applicant a_i that was matched with an organization higher than their worst achievable organization. Since this is organization proposing assume that no applicant has yet rejected an organization that is achievable for him. This means that no a_i has rejected any o_j where there exists a stable, slot stable match with a_i matched to o_j. If a_i rejected an organization for being unacceptable, it's unachievable. If a_i rejected o_u in favor of o_j, we know that the organization o_j has the applicant in their optimal list except for those that already rejected them and by the inductive assumption those applicants are unachievable to o_j. If we consider a hypothetical matching that matches a_i to o_u and everyone else to an achievable organization, a_i would prefer o_j and o_j would prefer a_i over at least one other a_k from o_j's more constrained optimal list making it an unstable match which violates Theorem 2.1. Q.E.D

Appendix C: Python ENPAP Code

```
import csv

'''
```

```
Reads in the applicant or organizational preferences from a csv file.
Expects each line to start with a unique applicant or organization name,
   followed by their ordered list of preferences.
'''
def __read_csv_with_row_header(file_path):
    rows = {}
    with open(file_path, 'r') as data:
        for line in csv.reader(data):
            # Need to account for duplicate names
            row_name = line[0]
            if row_name in rows:
                raise Exception(f'The file {file_path} has a duplicate row
                    header of {row_name}. Row headers must be unique')
            rows[row_name] = line[1:]
    return rows

def read_applicant_preferences(file_path):
    return __read_csv_with_row_header(file_path)

def read_organization_preferences(file_path):
    return __read_csv_with_row_header(file_path)

def applicant_fits(applicant, org_preferences, assigned_applicants_count):
    if assigned_applicants_count==0:
        return True
    else:
        return org_preferences.index(applicant) < org_preferences.index(
            str(assigned_applicants_count))

'''
Finds the applicant with the lowest preferred weight (Higher index=worse
   candidate)
'''
def find_lowest_ranked_applicant(applicant_weights):
    return max(applicant_weights, key=applicant_weights.get)

'''
Adding new applicants changes the marginal cost to the organization. This
   method
Looks for applicants that do not fit given the new marginal cost.
returns: map of applicant name to their index in preference order (higher
   is worse)
'''
def check_for_drops(org_preferences, current_assignments):
    assigned_applicant_count = len(current_assignments)
    potential_drops = {}
    #Check if no one would fit in new marginal cost
    if str(assigned_applicant_count) not in org_preferences:
        for assigned_applicant in current_assignments:
            potential_drops.update({assigned_applicant: org_preferences.
                index(assigned_applicant)})
    else:
        for assigned_applicant in current_assignments:
            # Check if any applicants already match fail new marginal cost
```

```
            if org_preferences.index(assigned_applicant) > org_preferences
                .index(str(assigned_applicant_count)):
                #Add them to a potential kick list
                potential_drops.update({assigned_applicant:
                    org_preferences.index(assigned_applicant)})
    return potential_drops

def enpap_algorithm(applicant_preference, organization_preferences):
    remaining_applicants = list(applicant_preference.keys())
    organization_assignments = {org : [] for org in
        organization_preferences.keys()}

    while(len(remaining_applicants)>0):
        current_applicant = remaining_applicants[0]
        current_preferences = applicant_preference[current_applicant]

        if len(current_preferences) == 0:           # if the applicant is
            out of preferences remove them
            remaining_applicants.pop(0)
        else:
            preferred_org = current_preferences[0]
            org_assignment_count = len(organization_assignments[
                preferred_org])
            org_preferences = organization_preferences[preferred_org]

            # Check if applicants fits
            if applicant_fits(current_applicant, org_preferences,
                org_assignment_count):
                #Add applicant to list
                organization_assignments[preferred_org].append(
                    current_applicant)

                potential_drops = check_for_drops(org_preferences,
                    organization_assignments[preferred_org])

                # remove applicant if necessary
                if len(potential_drops) > 0:
                    lowest_ranked = find_lowest_ranked_applicant(
                        potential_drops)
                    organization_assignments[preferred_org].remove(
                        lowest_ranked)
                    if lowest_ranked==current_applicant:
                        current_preferences.pop(0)
                    remaining_applicants.append(lowest_ranked)

                remaining_applicants.pop(0)

            else:
                current_preferences.pop(0)

    return organization_assignments


if __name__ == '__main__':
```

```
    applicant_preferences = read_applicant_preferences('appROL.csv')
    organization_preferences = read_organization_preferences('orgROL.csv')
    print(enpap_algorithm(applicant_preferences, organization_preferences)
       )
```