卒業研究アドバイザー選択制度の再設計:
Deferred Acceptance アルゴリズムの応用に関する実験的検討

The Redesign of a Thesis Advisor Assignment System: An Experimental Investigation on the Application of Deferred Acceptance Algorithm

染　谷　尚　希

SOMEYA, Naoki

181410

March, 2019

卒業研究アドバイザー選択制度の再設計:
Deferred Acceptance アルゴリズムの応用に関する実験的検討

The Redesign of a Thesis Advisor Assignment System:
An Experimental Investigation on the Application of
Deferred Acceptance Algorithm

English abstract appended

国際基督教大学教授会提出学士論文

A Thesis Presented to the Faculty of
the International Christian University
for the Baccalaureate Degree

by

染　谷　尚　希
SOMEYA, Naoki
181410

March, 2019

Approved by _____

金澤　雄一郎
論文指導審査委員
Thesis Advisor

目　　次

第 1 章　序論

1.1　研究の背景

私たちの身の回りにはマッチング現象が溢れている。例えば、結婚は人と人、商品購買は人とモノ、そして就職活動は人と組織のマッチングである。またその形態についても、例えば結婚は 1 対 1、プロ野球ドラフト会議は 1 対多、そして卸売業者と小売業者の取引関係は多対多のマッチングと言える。このように、身近なライフイベントからビジネスに至るまで、日常生活における様々な場面でマッチングは行われている。しかも、誰と、何とマッチングするかによって、プロジェクトの成否や生活の満足度、そして私たちの人生までもが影響を受ける。このように、マッチングは私たちにとって身近かつ重要な問題である。

マッチング理論とは、そのようなマッチング現象において、様々な選好を持つプレーヤー同士でどのようにペアを作り、限られた資源をどのように配分するかという問題を、ゲーム理論の考え方を応用して扱う数学的理論である。中でも、異なる 2 グループ間の 1 対 1 のマッチング現象において、それぞれのプレーヤーが相手グループのプレーヤーに対する選好順序のリストを持つ時、そのランキング情報に従ってどのような基準でペアを作れば、実現可能な中で互いに最も選好順序の高い組み合わせを得られるかという問題を、男女の結婚に例えて言及したのが、Gale and Shapley (1962) によって提示された安定結婚問題である。また、その際に用いるペアの決定プロセスが Deferred Acceptance Algorithm (DA アルゴリズム[1]) である。DA アルゴリズムは 1 対 1 だけでなく 1 対多、多々多のマッチング現象にも応用可能である。そして、それによって導出されるマッチングは、名称の由来である「受け入れを保留する」機能により、安定性と耐戦略性という性質を持つ。以下ではその 2 つの性質に、DA アルゴリズムにさらに期待される効果である申請者側最適性と停止性を加えた 4 つの性質について定義を確認する。

定義 1.1.1　安定性(stability)：あるマッチングが不安定であるというのは、その組み合わせの中に、ペアになっていない 2 人で、お互い今の相手よりも選好順序が高いペ

[1] 開発者の Gale と Shapley の名前に因んで Gale-Shapley アルゴリズムとも呼ばれる。アルゴリズムの具体的なプロセスや性質の分析については付録 A で詳しく解説する。

ア、つまりブロッキング・ペアが存在する場合である(Gale and Shapley, 1962)。よって、そのようなブロッキング・ペアが存在しないマッチングを安定マッチングという。また、異なる 2 グループ間のマッチングにおいて、安定マッチングは常に 1 つ以上存在し、その内の 1 つを DA アルゴリズムによって見つけることができる。

定義 1.1.2　耐戦略性(strategy-proof)：耐戦略性とは、マッチングにおいて他の参加者の戦略的な選好順序の操作に関わらず、参加者全員が各々の本来の選好順序に基づいて正直に申告することが支配戦略になる性質のことである。ただし、DA アルゴリズムが耐戦略性を常に持つのは申請者側のみであり、受入側は選好順序を操作することによって効用を高められる可能性がある。よって、常に安定マッチングを導出し、かつ、全ての参加者に対して耐戦略性を持つアルゴリズムは存在しない(Roth, 2008)。また、あるマッチングが耐戦略性を持つとき、そのマッチングはパレート最適である(Roth, 2008)。

定義 1.1.3　申請側最適性：DA アルゴリズムによって導出される安定マッチングは、申請側にとって常に最適である(Roth, 2008)。ナッシュ均衡として起こりうるマッチングのパターンの中で、例えば男性からプロポーズした場合は全ての男性にとって最適なマッチングが得られ、そのマッチングは女性にとって最悪となる。反対に女性からプロポーズした場合は全ての女性にとって最適なマッチングが得られ、そのマッチングは全ての男性にとって最悪となる。このように男性と女性、つまり申請側と受入側との間でトレードオフの関係になっている。

定義 1.1.4　停止性：DA アルゴリズムは有限回数で必ず停止する。n 人の男性がそれぞれ k 人の女性全員に対する選好順序のリストを持ち、それに従ってプロポーズしていくとする。男性は同じ女性に 2 度以上プロポーズしないことと、女性は一度婚約したら独身には戻らないことから、プロポーズという行為が全体で最大でも n*k 回行われた時点でアルゴリズムは終了する。よって計算量は O(n*k)である。

DA アルゴリズムはその有用性から理論と実践の両面で発展している。特に 1 対多のマッチングでは、Abdulkadiroğlu and Sönmez(2003)が学校選択制度への応用可能性

を明らかにし、ニューヨーク市とボストン市では既存の学校選択制度の分析と DA アルゴリズムを活用した新制度への提言がまとめられ(Abdulkadiroğlu et al, 2005a, Abdulkadiroğlu et al, 2005b)、著者らの提言に基づいて実際にニューヨーク市は 2003 年、ボストン市は 2005 年に制度改革が行われている。また、Roth and Peranson(1999) は既に DA アルゴリズムをベースに開発されていた研修医と病院のマッチングを図る全国医学実習生研修医マッチングプログラム(NRMP)に対して、既婚の研修医のパートナーと同じ病院に勤務したいという選好を踏まえた安定マッチングの開発に取り組んだ。この研修医マッチングプログラムは日本においても正式に実用化されている。またその他にも、Roth は腎臓移植の交換プログラムのアルゴリズム策定に寄与しており、安定マッチング理論と市場設計に関する功績を称えられ、2012 年に Shapley と共にノーベル経済学賞を受賞している。

1.2 研究の目的

マッチング理論は、経済学、特にゲーム理論で得られた知見を活かして現実の経済制度の設計・修正を行うマーケット・デザインという研究分野の一部である。これまでの伝統的な経済学が完全競争市場を想定し、解決策は市場の均衡に任せるという受身的な学問であったのに対し、マーケット・デザインは経済市場を設計・修正可能であると捉え、現実には存在しない完全競争市場ではなく、個々に異なる市場の特性に応じたルール設計を通じて、現実の市場を上手く機能させようとする能動的な学問である。実際に、経済学者が設計した仕組みが現実に応用される事例が増えてきており、学問としての成果が現実社会での価値に直結することから近年、より一層注目を集めている。

しかし、マッチング理論の知見はまだ現実に十分に活かされているとは言えない。というのも、今日において、特に 1 対多のマッチングにおけるペアの決定方法としてはボストン・メカニズム[2]という方法が採用されることが多い。ボストン・メカニズムは、DA アルゴリズムのようにステップ毎に受け入れを保留するのではなく、プレーヤーの選好順序を優先してステップ毎にペアを決定するため、必ずしも安定性や耐戦略性といった性質を持たない。理論的には、ボストン・メカニズムによって導出されるマッチングは DA アルゴリズムによって導出されるマッチングによってパレート支配される。と

[2] ボストン・メカニズムという名称の由来は、この方法が以前、ボストン市の学校選択制度で採用されていたことによる。アルゴリズムの具体的なプロセスは第付録 A で詳しく解説する。

ころが、仕組みが直感的でわかりやすいため、大学や自治体などで幅広く利用されている。実際に、国際基督教大学(ICU)の現行の卒業研究アドバイザー選択制度はボストン・メカニズムと同様のものである。そのため、不安定性と戦略的操作可能性という 2 つの問題が生じる可能性がある。

不安定性：現行の卒業研究アドバイザー選択制度では、例えば学生 s は 1 回目のマッチングで第 1 希望の教員とマッチできず、2 回目のマッチングで既に第 2 希望の教員 p の定員が上限に達していた場合、第 3 希望以下の教員にしか申請できず、結果的に第 3 希望の教員とマッチしたとする。この時、もし教員 p の学生 s に対する選好順序が、教員 p が結果的に受け入れた他の学生に対する選好順序よりも高かった場合、学生 s と教員 p はブロッキング・ペアとなる。よって、マッチング結果に正当な不満を持つペアが生じてしまう。

戦略的操作可能性：現行の卒業研究アドバイザー選択制度では 2019 年度から、学生は申請期間中に、その時点で各教員に何人の学生が申請しているのかを Web サイト上で一覧で確認することができる。この仕組みは教員毎の学生の受け入れ人数のばらつきを均し、一部の教員の負担を減らす施策としては有効であるかもしれない。しかし、申請者数を可視化することによって、例えば学生 s は 1 回目のマッチングで選好順序の高い教員となるべく確実にマッチしたいがために、あえて最も選好順序の高い教員 p ではなく、他の申請者数が少なくより確実にマッチできそうな第 2 希望以下の教員を第 1 希望として申請するかもしれない。もし教員 p の学生 s に対する選好順序が、教員 p が結果的に受け入れた他の学生に対する選好順序よりも高かった場合、本来成立したはずの互いに望ましいペアが生まれないという意味で機会損失であると捉えられる。

ICU の学生に対して実施したアンケート調査[3] によると、「下記に当てはまりますか。本当はより希望度の高い教員が他にいたが、その教員を希望する学生数が多かったため、

[3] 本研究では、ICU における卒業研究アドバイザー選択制度の現状をより正確に把握するために、学生に対するアンケート調査を行った。調査実施期間は 2019 年 1 月 22 日から同年 1 月 26 日である。調査対象は卒業研究アドバイザー選択制度を経験したことのある ID20(2019 年現時点において第 3 学年)以上の ICU 生で、合計 70 名から有効な回答を得られた。このアンケート調査で得られたデータを、制度設計を検討する際の定量的な根拠として逐次利用する。

できるだけ希望度が高い教員により確実に卒業研究アドバイザーになってもらうために、あえて別の教員の名前を希望調査リストの第 1 希望欄に書き込んだ。」という質問項目に対して「当てはまる」と回答した学生は全体の 8.6%であった。さらに、アンケート対象を 2019 年度以降の学生である ID20 のみに絞るとその割合は 20.0%であった。この数値自体に対する評価は、同様のケースでの比較材料を得られなかったため、本論文では言及できない。また、その内どれほどの割合が機会損失に繋がったのかについても、このアンケート結果からでは把握できない。ただ一方で、現行の卒業研究アドバイザー選択制度下においては戦略的に自身の選好順序を操作する学生が存在することは分かった。

また、戦略的選好への懸念に対して、申請者数が多いから諦めるという時点でその学生の最も選好順序の高かった教員に対するモチベーションはその程度なのでは、という意見も理解できる。しかしながら、大学での研究を重要視する学生にとっては、どの教員がアドバイザーになるかは非常に重大な問題であり、だからこそ最も希望する教員に対する他の学生の応募人数が多い場合、なるべく選好順序の高いゼミに所属するために戦略的に自らの希望順位を操作してしまうのは考慮できる。よって、そもそも選好順序の戦略的な操作が必要ない制度を設計することによって、学生が安心して自らの選好順序を正直に申告し、マッチング可能な中で最も選好順序の高い教員とマッチできるような仕組みづくりを検討する方が優先であると考える。

そこで本研究では、ICU の現行の卒業研究アドバイザー選択制度と比較する形で、DA アルゴリズムを活用した新しい制度設計について検討する。具体的には、ICU の個別具体的な状況設定を加味したペア決定プロセスを反映したプログラムをボストン・メカニズムと DA アルゴリズムの両方で作成し、コンピュータ上でシミュレーション実験を行う。また、制度が実際に機能するためには、安定性と耐戦略性を考慮するだけでは不十分で、学生と教員それぞれのマッチした相手に対する選好順序や、学生間の格差、各教員が用意しなくてはならない学生に対する選考順序のリストの数、そして全学生がいずれかの教員とマッチするまでのマッチング回数などといった変数が、現実的に実現可能な範囲に収まるようなアルゴリズムを設計しなくてはならない。そこで、アルゴリズムの実用性を評価するために、安定性と耐戦略性に加え、学生の効用・教員の効用・学生間の格差・教員の負担・実現可能性という合計 7 つの評価軸を設定し、多様な観点から比較する。本論文は、大学の卒業研究アドバイザー選択制度における DA アルゴリ

ズムの実用化について、現実的な制約を考慮した制度設計と、実際の状況をシミュレートしたコンピュータ実験による効果検証を通じて検討した事例として価値のある研究である。

1.3　本論文の構成

本論文では、第 2 章においてシミュレーション実験の方法について解説する。まず ICU の卒業研究アドバイザー選択制度の現状を確認し、実験を行うにあたって前提となる条件設定を行う。さらに、実験に用いるデータの生成方法、提案するアルゴリズムのプロセス、評価軸設定まで詳細に説明する。第 3 章では、シミュレーション実験の結果を提示する。ボストン・メカニズムと DA アルゴリズムを前章で設定した評価軸に従って比較する。そして第 4 章では、シミュレーション実験の結果をもとに DA アルゴリズムによる卒業研究アドバイザー選択制度について考察し、総括的な纏めとする。

第２章　シミュレーション実験の方法

本章では、シミュレーション実験の詳細な手法について説明する。まず、ICU における卒業研究アドバイザー選択制度の現状を概観し、それをもとに公平性に基づいたルール設定を行う。次に、コンピューター上でのシミュレーション実験に用いる選好順序のデータの生成方法と、検証する２種類のアルゴリズムについて説明する。最後に、それらのアルゴリズムを客観的に比較するための７つの評価軸を設定する。

2.1　ICU の卒業研究アドバイザー選択制度の現状分析

ICU に存在する学科は教養学部アーツ・サイエンス学科の 1 つのみで、その中に文理合わせて合計 31 の専攻科目が存在する。学生の専攻タイプはメジャー、メジャー・マイナー、ダブルメジャーの３種類あり、The ICU 学報 No.43[4]によると 2012 年３月から 2018 年 6 月までの 10 年間の卒業生数はそれぞれの専攻タイプ毎に 3,240 人、745 人、99 人である。ダブルメジャーの学生の中には卒業論文を２冊執筆する者もいるかもしれないが、原則として全ての専攻タイプで卒業要件となる卒業研究論文数は 1 冊である。また、ICU の学生向け WEB サイトにおける Registration Website の「教員別卒論アドヴァイザー申請者数」によると、2019 年 1 月 23 日時点で、アドバイザー申請済みの学生数は４月入学の学生と９月入学の学生合わせて 691 人であるのに対し、アドバイザー資格のある教員は 102 人であった。以上のデータから、2019 年度の卒業研究アドバイザー選択制度は、学生 691 人と教員 102 人の 1 対多マッチングであると考えられる。

ICU の卒業研究アドバイザー選択制度は基本的に 3 年次の冬学期に教務課の監督のもと行わる。そして学生は 4 年次の春から 1 年間、卒業研究アドバイザーの指導のもと卒業研究を進める。学生の中には 3 年次の春学期など早い時期から特定の教員に従いて専門的な研究を始める者もいるが、正式な卒業研究アドバイザー決定は 3 年次冬学期に一律に行われる。

ICU の卒業研究アドバイザー選択制度は以下のようなプロセスで行われる。まず大学の教務課が学内 Web サイトにて学生に対し卒業研究アドバイザー選択制度の申請期

[4] The ICU 学報 No.43. https://www.icu.ac.jp/about/docs/TheICU_No43.pdf, (参照 2019-01-28)

間を告知する。学生はその期間中に希望する教員との面談や情報収集を通じて、アドバイザーとして希望する教員を検討し、締め切りまでに学内 Web サイトの「卒論アドヴァイザー申請および卒業研究開始資格認定申請」ページにて上位第 3 希望までの教員名を入力する。この申請に対し、教員が学生の指導可能性を検討し、受け入れる学生を決定する。そのリストを最終的に教務課が確認することによって第 1 回目のマッチングが完了する。そして第 1 希望のゼミに所属できなかった学生に対して教務課が改めて希望教員の検討を要請し、学生はまだ定員の上限が満たされていない教員の中から希望する教員を検討し個別にマッチングを続ける。そして卒論アドバイザーを希望する全ての学生がいずれかの教員とマッチできた時点で、その年度の卒業研究アドバイザー選択制度は完了となる。よって、ICU の現行の卒業研究アドバイザー選択制度は、学生数 691 人、教員数 102 人、学生側申請かつ、学生の教員に対する選好順序のリストが 3 人のボストン・メカニズムによるマッチングであると考えられる。

2.2　シミュレーション実験のルール設定

本節では、卒業研究アドバイザー選択制度の現状を踏まえて、シミュレーション実験を行う上で必要な変数の値や範囲を設定し、また公平性に基づいた条件設定を行う。

・学生数：691 人

ICU には大学院が併設されており、教員は学部生と同時に大学院生の指導を行うが、本実験における対象は学部生のみに限定する。また、ICU は学際的な教育を推進しており、実際にアンケート結果によると「Q6. あなたのメジャーと、卒論アドバイザーの専門学科・研究領域は一致しましたか。」という質問項目に対し、「一致しなかった」と回答した学生は全体の 21.4%であった。以上のことから、卒業研究アドバイザー選択制度は専攻科目毎に分けるのではなく、全ての専攻科目を一律に対象とするのが望ましいと考える。よってシミュレーション実験において生成する学生数は 691 人と設定する。

・教員数：102 人

教員についても学生と同様に全ての専攻科目を範囲とするので、卒業研究アドバイザーとして学生を研究室に受け入れる資格のある全ての教員を対象とする。よってシミュレーション実験において生成する教員数は 102 人と設定する。

・学生の教員に対する選好順序のリスト数：3 人

学生が全ての教員に対して選好順序を付けるのは現実的に困難であり、また実質的に必要性がない。また、現行の卒業研究アドバイザー申請において求められる希望調査票が第 3 希望までであることから、学生は 1 回のマッチング試行で第 3 希望までのゼミを選択するものとする。もし 1 回のマッチング試行で卒業研究アドバイザーが決定していない学生が存在する場合は、さらに選好順序リストを第 6 位まで用意しマッチング試行を再実施する。

・教員の学生に対する選好順序のリストに載せる数：教員により可変

前提として、教員は自身のアドバイジー以外の学生の成績を確認することはできない。また、アンケート結果によると「卒論アドバイザー申請の際に、教員から以下のものについて提出するよう言われましたか。」という質問項目に対し、「成績」と答えた学生は 1.4%で、96.2%の学生は「何もなかった」と回答した。以上のことから、教員は学生に対する指導可能性を検討する際、成績のように全学生を線形に順位づけられる指標は使わずに、学生の興味関心・これまで履修した授業・面談等を通じて感じた相性などを総合的に考慮し検討するものとする。よって、各教員は 691 人の学生全員に対して選好順序のリストを作成するのは不可能であると仮定する。そこで、現実的な方法として、教員は実際に面談等を通じて教員へのアドバイザー担当の希望の意向を示した学生全員に対し選好順序のリストを作るものと設定する。そうすることで、教員はマッチする可能性のある学生に絞って選好順序のリストを作成すれば必要十分になり、そのリスト数も現実的な範囲内に抑えることができる。よって、教員の学生に対する選好順序のリストに載せる数は教員により可変である。

・1 人の教員が受け入れる学生数の上限：19 人

ICU の IR Office により提供されたデータによると、2018 年度において各教員がアドバイジーとして受け持つ学生数の一覧は以下の図 2.2.1 のような分布となる。なお、学生は学部生に限定している。本データから、最も所属学生数が多い教員 ID102 の 34 人と教員 ID101 の 25 人は外れ値として捉え、現実的な教員 1 人あたりの学生の受け入れ上限数は一律に 19 人と設定する。

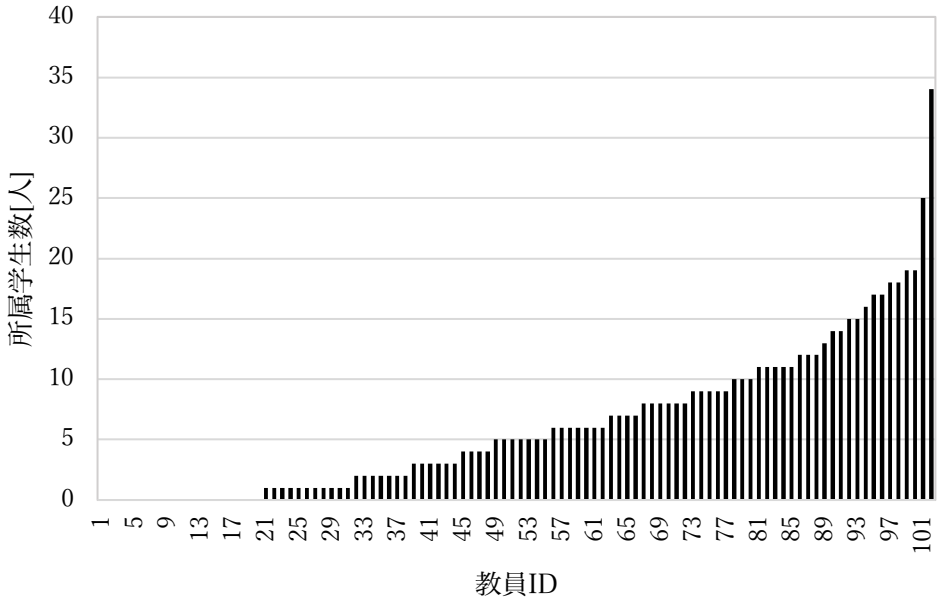

図 2.2.1：2018 年度における各教員が受け持つ学生数(学部限定)の一覧

・専攻科目による優遇の有無：無

アンケート結果によると「Q6. あなたのメジャーと、卒論アドバイザーの専門学科・研究領域は一致しましたか。」という質問項目に対し、「一致しなかった」と回答した学生は全体の 21.4%であったことから、必ずしも学生の専攻科目と教員の専門学域が一致するわけではない。また学生によっては分野横断的なテーマである等、必ずしも専攻科目と卒業研究テーマが連動するわけでもない。よって、自身の専攻科目と希望する教員の専攻学域が一致する学生の方が、一致しない学生よりもマッチングにおいて優先されるということはない、と設定する。

・マッチング回数：3 回以内

最初に学生が用意する選好順序のリストが第 3 希望までと制限される場合、初回のマッチングで全ての学生が教員とマッチできない可能性がある。一方で、学生と教員に対し何度も選好順序のリストを繰り返し作成させるのは大きな負担である。よって、実用可能性の高い制度を設計するために、マッチングが完了するまでの選好順序のリストの作成回数を、最大でも 3 回以内に収められるようなアルゴリズムが望ましいとする。

2.3　選好順序データの生成方法

・学生の教員に対する選好順序

学生の教員に対する選好順序の確率分布を、現実の分布とできるだけ近い確率で生成するために、ICU の学生向け Web サイトである Registration Website の「教員別卒論アドヴァイザー申請者数」で公開されている各教員への学生の応募者数を集計したものを用いた。データの分布は以下の図 2.3.1 に示される。

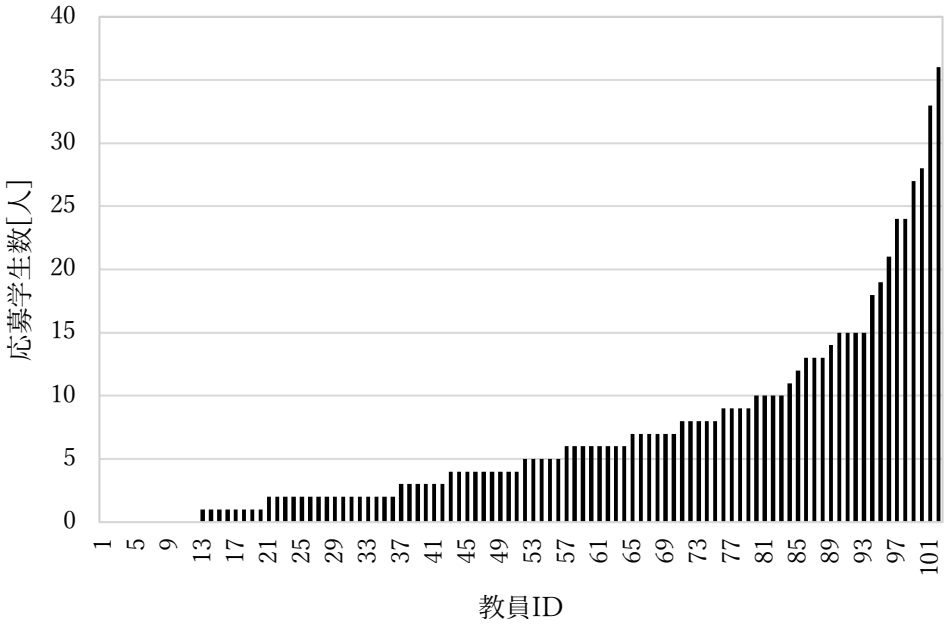

図 2.3.1：2019 年度における各教員への学生の応募者数の一覧

選好順序データの生成方法としては、各教員に対する学生応募者数を全て足し合わせた数を分母、各教員に対する個別の学生応募者数を分子に置くことで各教員が学生によって選択される確率を設定し、上記の確率でランダムに希望教員を選択することで選好順序データを生成した。ただし、この選好順序リストの生成方法には問題がある。というのも、使用したデータは既存の卒業研究アドバイザー選択制度の下で実際に生じたものであり、DA アルゴリズムの持つ耐戦略性、つまり選好順序を正直に申告することが最適戦略になることを事前に学生は理解していない状態で生成されたデータである。よって、今回の実験で用いる確率分布には学生の戦略的選好の影響が含まれている可能性がある。しかし、本実験においてはこの分布を、学生が自身の真の選好順序に従って正直に申告したものと仮定して実験を行う。

・教員の学生に対する選好順序

教員の学生に対する選好順序のリストを作成するにあたって、各学生が選択される確率にばらつきを発生させるために、任意の確率分布を作成した。具体的には、実験における各学生に、分子にその学生のインデックス番号の平方根かつ小数点以下を切り捨てた値をとり、分母にそれらの値の学生全員分の合計を置く形で確率分布を作成した。この確率分布によって生成されるデータの分布は以下の図 2.3.2 に示される。

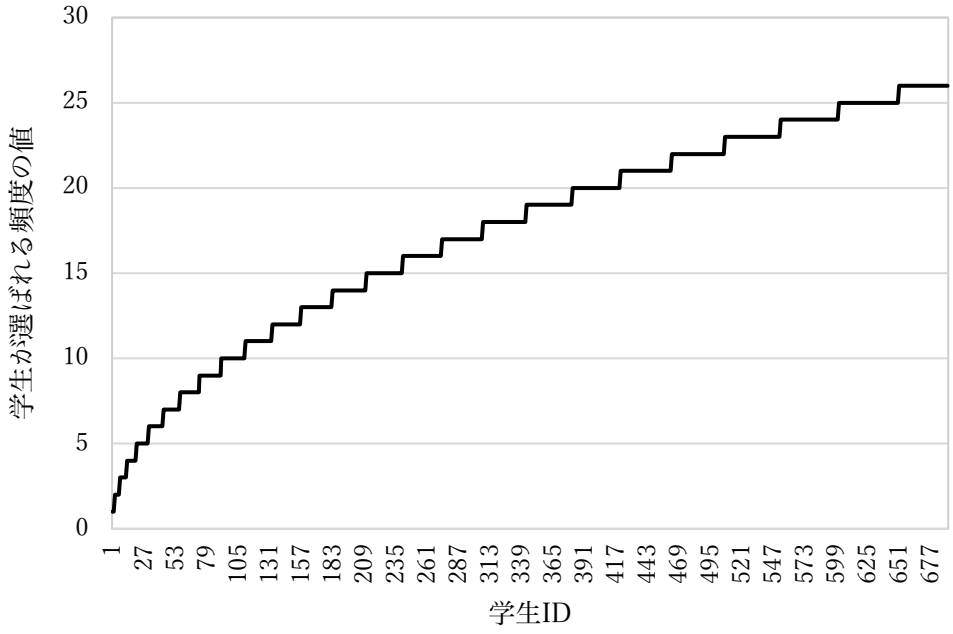

図 2.3.2：2019 年度における各教員への学生の応募者数の一覧

2.4　アルゴリズムの実装

本節ではシミュレーション実験で実行する 2 種類のアルゴリズムについて解説する。1 つは現行の卒業研究アドバイザー選択制度を反映したボストン・メカニズム、2 つ目は本論文で新しい制度として提案する DA アルゴリズムである。

本実験において、2 つのアルゴリズムは Python 3 で書いた。ライブラリは NumPy[5]・Pandas[6]・Matplotlib[7]をそれぞれデータの生成と計算・データの表示・グラフ描画のために使用した。プログラミング言語として Python 3 を採用した理由としては、文法がシンプルで実装しやすい点、科学計算のためのライブラリが豊富に存在する点、そしてアプリケーションへの実装がしやすい点の 3 つが挙げられる。また、アルゴリズムのプログラミングコードについては、ボストン・メカニズムは Delena Malen[8]、DA アルゴリズムは Miswar[9]のコードをベースに、ICU の卒業研究アドバイザー選択制度で利用できるよう機能を追加・修正する形で実装した。両プログラム共に追加した機能は主に以下の 2 つである。

まず第 1 に、全ての学生が教員とマッチするまで繰り返しマッチング処理を実行する機能を追加した。デフォルトのコードでは、今回のように学生の選好順序のリストが第 3 希望までという制約のもとでは、全ての学生が教員とマッチする前にアルゴリズムが停止してしまい、どの教員ともマッチできない学生が生じてしまう可能性があった。前提として卒業研究アドバイザー選択制度において学生がどの教員ともマッチできないという状況は避けなければならない。そこで、1 回目のマッチングが完了した後に教員とマッチしていない学生が存在する場合は、それらの学生とその時点で定員の上限が満たされていない教員とで新たに選好順序のリストを作成し、再度マッチング処理を行うようにした。

第 2 に、実験デザインで設定した「教員は実際に面談等を通じてゼミへの所属希望の

---

[5] NumPy. http://www.numpy.org/, (参照 2019-01-05)

[6] Pandas. https://pandas.pydata.org/, (参照 2019-01-05)

[7] Matplotlib. https://matplotlib.org/,(参照 2019-01-05)

[8] Delena Malena.
https://github.com/delenamalan/school_choice_algorithms/blob/master/python_implementations/boston_mechanism.py, (参照 2019-01-05)

[9] Miswar. https://github.com/miswar-repo/python-nrmp/blob/master/nrmp-applicant.py, (参照 2019-01-05)

意向を示した学生全員に対し選好順序のリストを作る」という状況を再現するために、教員の学生に対する選好順序のリストの作成方法を工夫した。一般的なアルゴリズムでは事前に互いに対する選好順序を有しているが、今回のアルゴリズムではまず学生側が教員に対する選好順序のリストを作成し、次にそのリストから順序データを取り除いたものを教員側が確認することで、教員側は自身がどの学生に申請されるかを事前に把握した状態で、自身とマッチする可能性のある学生のみに限定して選好順序のリスト作成できるようにした。この機能により教員の負担を大幅に減らすことができる。

シミュレーション実験で実際に使用したコードは付録 B に掲載した。本節ではアルゴリズムのプロセスを自然言語で擬似的に示す。まず、現行の卒業研究アドバイザー選択制度を反映したボストン・メカニズムのプロセスは以下の通りである。

ボストン・メカニズム

入力：

学生数：691 人

教員数：102 人

学生の教員に対する選好順序のリスト：3 人

教員の学生に対する選好順序のリスト：可変

教員 1 人あたりの学生の受け入れ上限数(定員)：20 人

出力：

学生と教員のペア

ステップ 1.0 ：

a. 各学生は教員に対する選好順序のリストを第 3 希望まで作成する。

b. 各教員は自身を選好順序のリストに含めた学生全員に対する選好順序のリストを作成する。(どの学生からも選好順序のリストに含められなかった教員は学生に対する選好順序のリストを作成しない。)

ステップ 1.1 ：

a. 各学生は、自身の選好順序において第 1 希望の教員にオファーする。

b. 各教員は、自身にオファーした学生の中から選好順序が最も高い学生を受け入れる。全ての学生が教員とマッチするか、オファーする学生がいなくなった時点で割り当てを停止する。

ステップ 1.k(k≧2)：

a. まだ教員とマッチしていない各学生は、自身の選好順序において第 k 希望の教員にオファーする。(自身の選好順序のリストの中に定員が上限に達していない教員、または 1 度もオファーしていない教員がいなくなった学生はオファーをしない。)

b. 定員が上限に達していない各教員は、その教員にオファーした学生の中から選好順序が最も高い学生を受け入れる。全ての学生が教員とマッチするか、オファーする学生がいなくなった時点で割り当てを停止する。メカニズムの停止条件が満たされない限り、ステップ 1.k を繰り返す。

メカニズムの停止条件 1：

a. 新しいオファーがされなくなった時点でメカニズムを停止し、その時点で各教員が受け入れている学生をその教員とマッチさせることで 1 回目のマッチングを生成する。

b. まだ教員とマッチしていない学生が存在する場合、全ての学生が教員とマッチするまで以下のステップを繰り返す。

ステップ n.0 ：

a. まだ教員とマッチしていない各学生は教員に対する選好順序のリストを第 3+n 希望まで増やす。

b. 定員が上限に達していない各教員は、自身を選好順序のリストに含めた学生全員に対する選好順序のリストを作成する。(どの学生からも選好順序のリストに含められなかった教員は学生に対する選好順序のリストを作成しない。)

ステップ n.1：

a. 各学生は、自身の選好順序において第 p(1+3*n)希望の教員にオファーする。

b. 各教員は、自身にオファーした学生の中から選好順序が最も高い学生を受け入れる。全ての学生が教員とマッチするか、オファーする学生がいなくなった時点で割り当てを停止する。

ステップ n.k(k≧2)：

a. まだ教員とマッチしていない各学生は、自身の選好順序において第 q(k+3*n)希望の教員にオファーする。(自身の選好順序のリストの中に定員が上限に達していない教員、または 1 度もオファーしていない教員がいなくなった学生はオファーをしない。)

b. 定員が上限に達していない各教員は、その教員にオファーした学生の中から選好順序が最も高い学生を受け入れる。全ての学生が教員とマッチするか、オファーする学生がいなくなった時点で割り当てを停止する。メカニズムの停止条件が満たされない限り、ステップ n.k を繰り返す。

メカニズムの停止条件 n+1：

新しいオファーがされなくなった時点でメカニズムを停止し、その時点で各教員が受け入れている学生をその教員とマッチさせることで n 回目のマッチングを生成する。

次に、新しく提案する卒業研究アドバイザー選択制度を反映した DA アルゴリズムのプロセスは以下の通りである。

DA アルゴリズム

入力：

学生数：691 人

教員数：102 人

学生の教員に対する選好順序のリスト：3 人

教員の学生に対する選好順序のリスト：可変

教員 1 人あたりの学生の受け入れ上限数(定員)：20 人

出力：

学生と教員のペア

ステップ 1.0：

a. 各学生は教員に対する選好順序のリストを第 3 希望まで作成する。

b. 各教員は自身を選好順序のリストに含めた学生全員に対する選好順序のリストを作成する。(どの学生からも選好順序のリストに含められなかった教員は学生に対する選好順序のリストを作成しない。)

ステップ 1.1：

a. 各学生は、自身の選好順序において第 1 希望の教員にオファーする。

b. 各教員は、自身にオファーした学生の中から、選好順序が最も高い学生を定員上限まで一時的に確保し、それ以外を断る。オファーした学生数が定員未満なら、その学生全員を一時的に確保する。

ステップ 1.k(k≧2)：

c. 前のステップで断られた学生は、まだ断られていない教員の中から選好順序において最も順位の高い教員にオファーする。(1 度もオファーしていない教員がいなくなった学生はオファーをしない。)

d. 各教員は、自身にオファーした学生とステップの時点で確保した学生の中から、自身の選好順序で最も順位が高い学生を割当上限まで一時的に確保し、それ以外を断る。メカニズムの停止条件が満たされない限り、ステップ 1.k を繰り返す。

メカニズムの停止条件 1：

新しいオファーがされなくなった時点でメカニズムを停止し、その時点で各教員が確保している学生をその教員とマッチさせることで 1 回目のマッチングを生成する。

まだ教員とマッチしていない学生が存在する場合、全ての学生が教員とマッチするまで以下のステップを繰り返す。

ステップ n.0：

a. まだ教員とマッチしていない各学生は教員に対する選好順序のリストを第 3+n 希望まで増やす。

b. 定員が上限に達していない各教員は、自身を選好順序のリストに含めた学生全員に対する選好順序のリストを作成する。(どの学生からも選好順序のリストに含められなかった教員は学生に対する選好順序を作成しない。)

ステップ n.1：

e. 各学生は、自身の選好順序において第 p(1+3*n)希望の教員にオファーする。

f. 各教員は、自身にオファーした学生の中から、選好順序が最も高い学生を定員上限まで一時的に確保し、それ以外を断る。オファーした学生数が定員未満なら、その学生全員を一時的に確保する。

ステップ n.k(k≧2)：

g. 前のステップで断られた学生は、まだ断られていない教員の中から選好順序において第 q(k+3*n)希望の教員にオファーする。(1 度もオファーしていない教員がいなくなった学生はオファーをしない。)

h. 各教員は、自身にオファーした学生とステップの時点で確保した学生の中から、自身の選好順序で最も順位が高い学生を割当上限まで一時的に確保し、それ以外を断る。メカニズムの停止条件が満たされない限り、ステップ 1.k を繰り返す。

b .メカニズムの停止条件 n+1：

新しいオファーがされなくなった時点でメカニズムを停止し、その時点で各教員が確保している学生をその教員とマッチさせることで n 回目のマッチングを生成する。

2.5　評価軸設定

ボストン・メカニズムによる既存の制度と新しく提案する DA アルゴリズムによる制度を多面的に比較するために、安定性・耐戦略性・学生の効用・教員の効用・学生間の格差・教員の負担・実現可能性という 7 つの評価軸を設定する。以下では、それぞれの意味と算出方法について説明する。

(a)安定性

制度の持続可能を測る評価軸として設定する。算出方法としては、導出されるマッチングにおけるブロッキング・ペアの数を求める。よって、この値が小さいほどブロッキング・ペアが少なく制度に対して不満を持つ者が少ないことを意味する。理論的には、DA アルゴリズムの場合、この評価値は 0 になることが予想される。

(b)耐戦略性

学生による戦略的選好の有効性を測る評価軸として設定する。算出方法としては、嘘の選好順序を申告することでより良い教員とマッチングできる学生のユニーク数を求める。よって、この評価値が小さいほど、選好順序を操作しても得する学生が少ないことを意味し、評価値が 0 の場合はその制度が耐戦略性を満たしていることを意味する。理論的には、DA アルゴリズムの場合、この評価値は 0 になることが予想される。ただし、この数値はあくまで戦略的操作が起こりうる余地であり、評価方法として厳密には正しくない。しかし、本実験では両アルゴリズムにおける戦略的操作が事前に発生する可能性の違いについて検証できなかったので、上述のような算出方法を採用した。

(c)学生の効用

学生がマッチした教員に対する満足度を測る評価軸として設定する。算出方法としては、各学生のマッチした教員に対する選好順序の平均値を求める。よって、この評価値が小さいほど、学生は希望する教員とマッチできていることを示す。アルゴリズムの仕組みから予想される結果としては、ボストン・メカニズムはステップ毎にペアを確定するのに対し、DA アルゴリズムは一度暫定的に組んだペアを後のステップで解消する可能性があるため、第 1 希望の教員とマッチする学生数に関してはボストン・メカニズムの方が DA アルゴリズムよりも多くなると考えられる

(d)教員の効用

教員がマッチした相手に対する満足度を測る評価軸として設定する。算出方法としては、各教員のマッチした学生に対する選好順序の平均値を求める。よって、この評価値が小さいほど、教員は希望する相手とマッチできていることを示す。

(e)学生間の格差

学生間における格差を測る評価軸として設定する。算出方法としては第 3 希望以下の教員とマッチした学生の人数を求める。この評価値が高いほど、学生間・ゼミ間の格差が大きいことを意味する。

(f)教員の負担

教員が学生に対し指導可能性を検討する上での負担を測る評価軸として設定する。算出方法としては、教員の学生に対する選好順序のリストの数の平均を求める。よって、この値が大きいほど教員は大人数の学生に対し指導可能性を検討する必要があり、負担が大きくなることを意味する。

(g)実現可能性

制度の実用化が可能かどうかを測る評価軸として設定する。実際の ICU の卒業研究アドバイザー選択制度を振り返ると、学生が面接等を通じて申請する教員を決定し第 3 希望までのリストを提出するまでの流れを 1 回のマッチングと捉えると、この回数が増えるほど手間が増えると考える。そこで算出方法としては、全ての学生が教員とマッチするまでのマッチング回数を求める。よって、この評価値が 3 回を越えると、学生・教員・教員にとって負担が増えるため、現実的に導入は困難であると考えられる。

第３章　シミュレーション実験の結果

全ての学生が自身の選好順序に従って正直に応募したと仮定して、それぞれのアルゴリズムによって試行回数 1000 回ずつコンピュータ上でシミュレーション実験を行った。各評価軸について平均値と標準偏差を算出すると、以下の表 3.1 のような結果が得られた。なお、数値は全て小数点以下第４位で四捨五入している。

n=1000	ボストン・メカニズム		DAアルゴリズム	
評価軸	平均	標準偏差	平均	標準偏差
(a)安定性	6.348	2.807	0	0
(b)耐戦略性	5.961	2.587	0	0
(c)学生の効用	1.134	0.453	1.138	0.395
(d)教員の効用	114.016	3.439	112.499	3.725
(e)学生間の格差	24.157	6.136	10.718	4.437
(f)教員の負担	20.558	21.881	20.367	21.853
(g)実現可能性	1.993	0.083	1.756	0.43
＊計算時間	3.010秒	-	2.708秒	-

表 3.1：シミュレーション実験の結果

まず(a)安定性については、ボストン・メカニズムでは平均 6.348 組のブロッキング・ペアが発生したのに対し、DA アルゴリズムでは理論通りブロッキング・ペアは 1 組も発生せず、安定性が証明された。よって、(a)安定性では DA アルゴリズムの方が優れている。ブロッキング・ペアの発生組数とその頻度は、以下の図 3.2 のヒストグラムに示されるような結果となった。

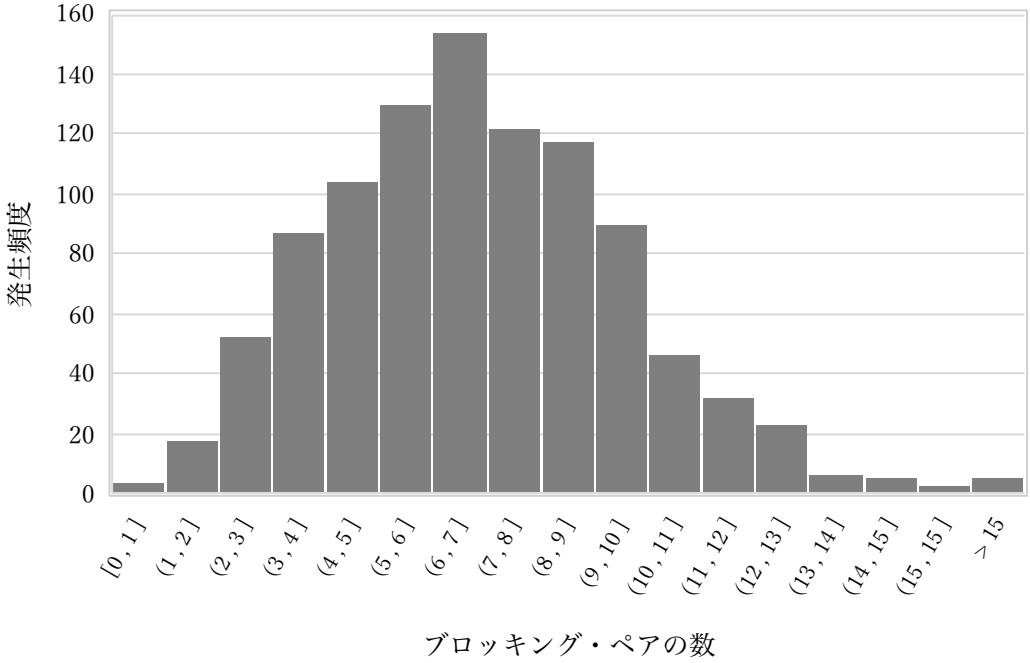

図 3.2：ブロッキング・ペアの発生組数とその頻度のヒストグラム

また(b)耐戦略性についても、ボストン・メカニズムでは選好順序を操作して得する学生が平均 4.931 人発生したのに対し、DA アルゴリズムでは理論通りそのような学生は 0 人であることから、学生側の耐戦略性が証明された。よって、(b)耐戦略性でも DA アルゴリズムの方が優れている。次に、(c)学生の効用と(d)教員の効用については、両者ともに平均値に差は見られなかった。ただ、(c)学生の効用の標準偏差については DA アルゴリズムの方が小さいことから、各学生がマッチした教員に対する選好順序のばらつきは DA アルゴリズムの方が少ないと考えられる。実際に以下の表 3.3、表 3.4 を見ると、両アルゴリズムの機能の違いから予測されたように、ボストン・メカニズムの方が第 1 希望の教員とマッチする学生の割合が高かった。一方で、DA アルゴリズムはボストン・メカニズムと比較して第 2 希望の教員とのマッチ率が高く、結果的に第 3 希望以下の選好順序の低い教員とのマッチ率が低かった。実際に、(e)学生間の格差を見ると、第 3 希望以下の教員とマッチした学生の割合がボストン・メカニズムでは平均 24.157 人であるに対し、DA アルゴリズムでは 10.718 人である。以上のことから、DA アルゴリズムは学生の第 1 希望の教員とのマッチ率は下げる代わりに、マッチした教員に対する選好順序の学生間における格差を減らすことがわかった。

n=1000	ボストン・メカニズム		DAアルゴリズム	
選好順序	人数	比率	人数	比率
第1希望	616.495	89.218%	607.595	87.930%
第2希望	50.348	7.286%	72.687	10.519%
第3希望	16.187	2.343%	9.248	1.338%
第4希望	7.939	1.149%	1.469	0.213%
第5希望	0.031	0.004%	0.001	0.0001%

表 3.3：教員に対する選好順序毎の学生のマッチ数と割合

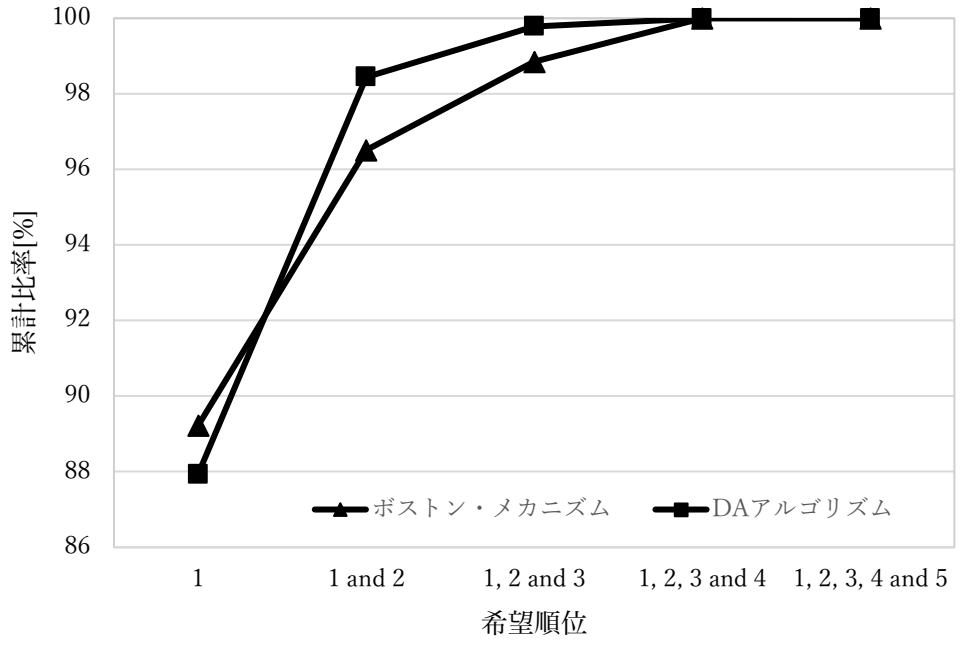

図 3.4：マッチした教員に対する選好順序毎の学生の割合の累計比率

続いて、(f)教員の負担については両者で差は見られなかった。ここで、各教員が選好順序のリストに含めた学生数の平均は両アルゴリズムともに約 20 人であるが、一方で標準偏差が非常に大きい。以下の図 3.5 を見ると明らかなように、教員によってリストに載せなければならない学生数は大きく異なる。50 人以上の学生に対して選好順序を検討しなければならない教員は全体の中でも 1 割程度であるが、比較的学生が多く集まりやすい教員は、平均を大きく超えた人数の学生に対して選好順序を検討しなければならないようだ。

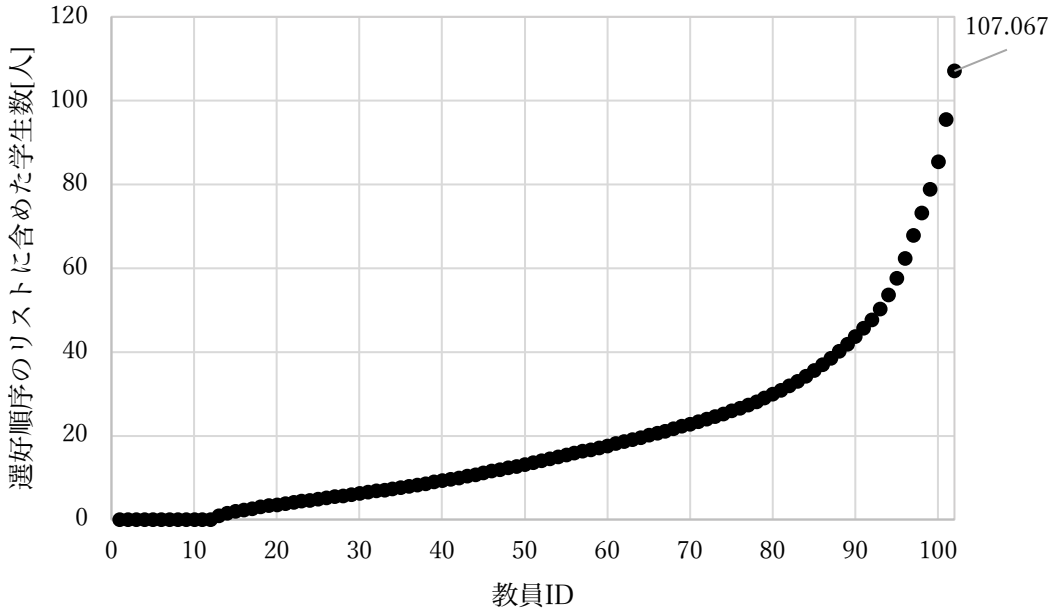

図 3.5：各教員が自身の選好順序のリストに含めた学生数の一覧

そして(g)実現可能性については、全ての学生が教員とマッチするまでのマッチング回数が、ボストン・メカニズムでは平均 1.993 回であるのに対し、DA アルゴリズムでは平均 1.756 回であることから、DA アルゴリズムの方が比較的少ない回数でマッチング処理が完了することがわかった。また、両者ともに 1000 回の試行回数の中で最大のマッチング回数は 2 回だったので、目安で設定した 3 回以内に収まった。

最後に、今回の実験で使用したアルゴリズムの試行回数 1 回分の平均計算時間としては、ボストン・メカニズムでは 3.01 秒、DA アルゴリズムでは 2.708 秒であった。

第４章　考察と結論

本章では、シミュレーション実験の結果をもとに、DA アルゴリズムによる卒業研究アドバイザー選択制度の妥当性について検討する。そして、最後に結論と共に研究の振り返りをして本論文を締めくくりたい。

4.1　考察

まず初めに、本実験の妥当性について考える。現行の制度を模したボストン・メカニズムによって得られた、学生の最終的にマッチした教員に対する選好順序の割合は第 1 位が 89.218%、第 2 位が 7.286%、第 3 位が 2.343%、それ以下が 1.153%であった。それに対し、学生に対するアンケート調査における「Q5. 最終的に第何希望の卒論アドバイザーに決定しましたか。」という質問項目に対する回答は、第 1 位が 89.2%、第 2 位が 7.0%、第 3 位が 2.8%であった。以上の数値を踏まえると、今回の実験デザインとしての各変数のルール設定や選好順序データのモデル設定は概ね妥当であったと評価できる。そこで以下では、シミュレーション実験の結果をもとに、DA アルゴリズムによる卒業研究アドバイザー選択制度について①安定性と耐戦略性、②学生と教員の効用、そして③制度としての実現可能性の 3 点に焦点を当てて考察を行う。

まず①安定性と耐戦略性に関しては、理論通り安定マッチングを導出し、学生に対する耐戦略性も発揮した。しかしながら厳密な数値として見ると、DA アルゴリズムによって発生を防ぐことができたブロッキング・ペアの数は平均で約 6 組かつ、戦略的選好を行う余地のある学生数は約 6 人であり、全体の 691 人という学生数から見ると比較的小さい数字である。このように安定性と耐戦略性の恩恵を受けた学生が少なかったのは、ICU の卒業研究アドバイザー選択制度自体が申請側である学生にとって”易しい”ゲームであったからだと考えられる。というのも、学生 691 人に対し、教員の枠は理論上 1938(=教員 102 人×教員 1 人あたりの定員 19 人)存在するため、学生にとって選好順序の高い教員とマッチすることは比較的容易であったと考えられる。ただし、本実験では定員を一律 19 人に設定したものの、現実には個々の教員によって異なるため、実際の枠はより少ない可能性がある。よって、学生だけでなく教員に対してもアンケート調査等を行い、個々の教員の受け入れ学生数の上限を把握できていれば、より正確な分析が可能だったかもしれない。いずれにせよ、本研究の限りでは、DA アルゴリズムの

安定性と耐戦略性の効果は発揮されたものの、制度変更に伴い他のデメリットが生じた場合に、そのデメリットをカバーできるほどのメリットを享受できるか懸念が残る。

次に、②学生と教員の効用に関しては、DA アルゴリズムによって全体的な効用が高まるわけではないが、学生間の効用の格差を抑えられることが明らかになった。第 1 希望の教員とマッチできる学生の割合の減少はデメリットと捉えられる面もあるが、その減少率は本実験結果によると 1.288%である。その数字が意味するところを、学生が抱く正統な不満の解消として解釈すると、逆にメリットとして捉えられるかもしれない。

最後に、③制度としての実現可能性に関して、DA アルゴリズムの導入は、本研究において考慮した変数の範囲内においては、現実的に可能であると考えられる。全ての学生が教員とマッチするまでのマッチング回数については、現行の制度を表すボストン・メカニズムよりも比較的少ない結果となった。さらに、教員が選好順序を考慮する必要のある学生数はボストン・メカニズムの場合と差が見られなかった。ただし、実際のマッチングの場面では、現行の制度下において学生が希望申請時に記入する第 2 希望と第 3 希望は、それ自体がマッチングに利用されるわけではなく、第 1 希望にマッチできなかった時点で改めて再検討を求められるので実質的にあまり意味を為さない。そのため、DA アルゴリズムを導入した場合、学生が希望申請時に記入する第 2 希望と第 3 希望はそれ自体がマッチングに利用されるため、教員としては学生に希望される頻度に変化はないが、以前よりも意味のある選好順序付けを行わなくてはならない。そういった点で、実用化には教員の負担が伴う可能性がある。

4.2 結論

本研究では、ICU の卒業研究アドバイザー選択制度における DA アルゴリズムの実用可能性を、コンピュータ上のシミュレーション実験によって検討した。そして以下の事項が明らかになった。まず第 1 に、安定性と耐戦略性を備えたマッチングの導出は可能であるが、ICU のように申請側の需要に対する受入側の供給が大きい場合、それによって実際に恩恵を受ける学生数は多いとは言えない。第 2 に、全体的な効用に影響は与えないが、学生間の効用の格差は縮小した。第 3 に、学生と教員が互いに意味のある選好順序のリストを作成できれば、DA アルゴリズムの導入は十分に実現可能である。

続いて、本研究における限界は主に 3 点ある。第 1 に、説明力のある選好順序データのモデルが作成できなかった。教員が学生から選ばれる際のデータモデルは現実の確率

分布に近いであろうものの、学生の戦略的選好の影響を排除できなかった。さらに、学生が教員から選ばれる際の確率分布については論理的な設計ができなかった。第 2 に、耐戦略性が実際にどれほど機能するかについては検証できなかった。たとえ配属方法に DA アルゴリズムを採用し、学生に対して事前に正直申告が最適戦略であることを説明したとしても、その意味を理解せず、もしくは効果を信頼せずに依然として選好順序を操作する学生は発生するであろう。そうした学生の行動傾向をさらに把握することで、制度設計に対するより緻密な分析が行えたかもしれない。第 3 に、学生が真に意味のある選好順序のリストを作成できるような仕組み作りは検討できなかった。アンケート調査の「卒論アドバイザーを検討する際、何人の教員と面談しましたか。」という質問に対する回答によると、全体の 65.7%の学生は 1 人の教員としか面談しておらず、2 人と面談したのは 22.9%、3 人以上と面談したのは 10%であった。教員と全く面談しなかった学生も 1 人存在した。DA アルゴリズムは有効な選好順序のリストをマッチング参加者が有することで最適なマッチングを導出する。しかし、面談人数のデータを見ると、学生の第 2、第 3 希望以下の選好順序は熟慮されていない、”とりあえず記入した”ものである可能性が高い。これではいくら配属制度を整備したところで本質的な意味を為さない。よって、本質的に最適な学生と教員のマッチングを目指すのであれば、マッチング以前の段階として、学生が研究に興味を持ち、自身に本当に適した教員を検討し発見できるような環境を同時に整備していく必要がある。この点に関しては、今後も様々なシチュエーションにおいて最適なマッチングの実現を模索する上での課題としたい。

最後に、本論文では最適なマッチングをいかに実現するか、ということを論じてきたが、一方でもちろん、運命的な出会いや、自身の知識や経験に縛られない選択といった偶然性が人生を豊かにすることも多い。何しろ、この世界は現時点の科学技術の下では不可逆であり、そもそも現実に起こり得なかったもう 1 つの選択について考え続けても仕方がない。偶然的にも選択した、導かれたその道が全てであり、もしその選択がネガティブに感じるのであれば、その経験を将来より良い選択ができるよう活かすより他無いであろう。しかしながら、少なくとも本人が、自身が何を望んでいるのかを理解し、目標を設定し、それを達成するのが善であると信じている場合、つまりその選択やマッチングの成否が知覚可能な範囲内においては、より希望する選択肢が本人にとって獲得されるべきである。そういったマッチングがより実現しやすい環境が理論的に設計可能であるならば、その知見を活かすべく積極的に効果を検証していく必要があると考える。

参考文献

Abdulkadiroğlu, A. and Sönmez, T. (2003). “School Choice: A Mechanism Design Approach,” *American Economic Review*, 93(3), pp.729-747.

Abdulkadiroğlu, A., Pathak, P. and Roth, A. (2005). “The New York City High School Match,” *American Economic Review*, 95(2), pp.364-367.

Abdulkadiroğlu, A., Pathak, P., Roth, A. and Sönmez, T. (2005). “The Boston Public School Matc,” *American Economic Review*, 95(2), pp.368-371.

Gale, D. and Shapley, L. (1962). “College Admissions the Stability of Marriage.” *American Mathematical Monthly*, Vol.69, pp.9-15.

Roth, A. E. (2008). “Deferred acceptance algorithms: History, theory, practice, and open questions,” *International Journal of game Theory*, 36(3-4):537–569.

Roth, A. and Peranson, E. (1999). “The Redesign of the Matching Market for American Physicians: Some Engineering Aspects of Economic Design,” *American Economic Review*, 89(4), pp.748-780.

付録 A　マッチング理論

DA アルゴリズムの構造を詳しく紹介するために、アルゴリズムのプロセス、安定性の証明、ケース分析についてボストン・メカニズムと比較しながら説明する。

A.1　アルゴリズムのプロセス

学生と教員のマッチングは基本的に 1 対多のマッチング現象であることが多いが、ここではアルゴリズムの基本構造を理解しやすくするために、学生と教員の 1 対 1 のマッチングを想定したアルゴリズムのプロセスを自然言語で示す。なお、以下のプロセスは両方とも学生側からオファーする形式に限定している。まず、現行の卒業研究アドバイザー選択制度と同等の原理で動くボストン・メカニズムのプロセスは以下の通りである。

ボストン・メカニズム

ステップ 0：

各教員及び学生はそれぞれ、相手グループ全員に対する選好順序を申告する。また初期設定として、全員誰ともマッチしていない状態とする。

ステップ 1：

a. 各学生は自身の選好順序において第 1 希望の教員にオファーする。
b. 各教員は自身にオファーした学生の中から選好順序が最も高い学生 1 人をアドバイジーとして受け入れ、他を拒否する。

ステップ k(k≧2)：

a. 前ステップで拒否された学生は、まだどの学生ともマッチしていない教員の中で最も自身の選好順序の高い教員を 1 人選び申請する。(自身の選好順序のリストの中にまだどの学生ともマッチしていない教員、または 1 度もオファーしていない教員がいなくなった学生はオファーをしない。)
b. どの学生ともマッチしていない教員は、自身にオファーした学生の中から選好順序が最も高い学生 1 人をアドバイジーとして受け入れ、他を拒否する。メカニズムの停止条件が満たされない限り、ステップ k を繰り返す。

メカニズムの停止条件：

全ての学生がいずれかの教員とマッチした時点でメカニズムを停止し、その時点でペアになっている学生と教員をマッチングとして生成する。

DA アルゴリズムは 1 つ以上の安定マッチングを導出する。また、出力される安定マッチングは学生がどのような順番でオファーするかによらない。プロセスは以下の通りである。

DA アルゴリズム

ステップ 0：

各教員及び学生はそれぞれ、相手グループ全員に対する選好順序を申告する。また初期設定として全員誰ともマッチしていない状態とする。

ステップ 1：

a. 各学生は自身の選好順序において第 1 希望の教員にオファーする。
b. 各教員は自身にオファーした学生の中から選好順序が最も高い学生 1 人をアドバイジーとして暫定的に確保し、他を拒否する。

ステップ k(k≧2)：

a. 前ステップで拒否された学生は、まだどの学生ともマッチしていない教員の中で最も自身の選好順序の高い教員を 1 人選び申請する。(自身の選好順序のリストの中にまだどの学生ともマッチしていない教員、または 1 度もオファーしていない教員がいなくなった学生はオファーをしない。)
b. どの学生ともマッチしていない教員は、自身にオファーした学生の中から選好順序が最も高い学生 1 人をアドバイジーとして暫定的に選び、他を拒否する。メカニズムの停止条件が満たされない限り、ステップ k を繰り返す。

メカニズムの停止条件：

全ての学生がいずれかの教員とマッチした時点でメカニズムを停止し、その時点で各教員が暫定的に確保している学生とのペアをマッチングとして生成する。

A.2　安定性の証明

安定性の定理は背理法によって証明できる。学生 s が最初にオファーを拒否された学生であるという前提のもとで、学生 s は教員 p と最終的にマッチしたとする。もしこのペア以外に安定した、より良い組み合わせがあると仮定すると、教員 p'の存在が想定されるが、学生 s にとって教員 p よりも教員 p'の選好順序が高かったとしても最終的に教員 p'に拒否されたとすれば、教員 p'にとって学生 s よりも選好順位の高い学生 s'の存在が想定される。もし学生 s と教員 p'が本当に安定的なペアであるならば、学生 s'には教員 p'よりも選好順序の高い教員 p''が存在したはずで、学生 s'が教員 p'とマッチするためには学生 s'は教員 p''に既に拒否されていなければならない。この時点で、学生 s'が学生 s よりも先に拒否されたという矛盾が生じるので、学生 s と教員 p のペアよりも安定的なペアは存在できないことが証明される。

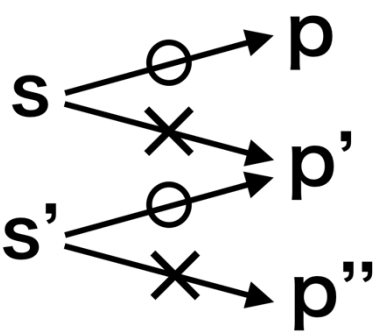

図 A.2. 1

A.3　ケース分析

DA アルゴリズムの持つ安定性・耐戦略性・申請者最適の 3 つの性質を確かめるために、表 A3.1、表 A3.2 のような選好順序のリストを持つ学生(s1, s2, s3, s4, s5)と教員(p1, p2, p3, p4, p5)のマッチングゲームを考察する。

学生	1位	2位	3位	4位	5位
s1	p3	p5	p1	p4	p2
s2	p1	p2	p3	p4	p5
s3	p1	p4	p3	p5	p2
s4	p2	p5	p4	p1	p3
s5	p4	p1	p5	p2	p3

表A3.1：学生の教員に対する選好順序

教員	1位	2位	3位	4位	5位
p1	s4	s1	s3	s2	s5
p2	s5	s1	s2	s3	s4
p3	s2	s3	s1	s5	s4
p4	s3	s4	s2	s5	s1
p5	s3	s1	s5	s2	s4

表A3.2：教員の学生に対する選好順序

上記の選好順序のリストに従って、①ボストン・メカニズム(学生側申請)、②DA アルゴリズム(学生側申請)、③DA アルゴリズム(教員側申請)の 3 種類のアルゴリズムを用いて、申請者側の正直申告を仮定したマッチングを行ったところ、①の結果は表 A.3.3 と表 A.3.4、②の結果は表 A.3.5 と表 A.3.6、③の結果は表 A.3.7 と表 A.3.8 のようになった。

学生	1位	2位	3位	4位	5位
s1	p3	p5	p1	p4	p2
s2	p1	p2	p3	p4	p5
s3	p1	p4	p3	p5	p2
s4	p2	p5	p4	p1	p3
s5	p4	p1	p5	p2	p3

表A3.3：①結果(学生視点)

教員	1位	2位	3位	4位	5位
p1	s4	s1	s3	s2	s5
p2	s5	s1	s2	s3	s4
p3	s2	s3	s1	s5	s4
p4	s3	s4	s2	s5	s1
p5	s3	s1	s5	s2	s4

表A3.4：①結果(教員視点)

学生	1位	2位	3位	4位	5位
s1	p3	p5	p1	p4	p2
s2	p1	p2	p3	p4	p5
s3	p1	p4	p3	p5	p2
s4	p2	p5	p4	p1	p3
s5	p4	p1	p5	p2	p3

表A3.5：②結果(学生視点)

教員	1位	2位	3位	4位	5位
p1	s4	s1	s3	s2	s5
p2	s5	s1	s2	s3	s4
p3	s2	s3	s1	s5	s4
p4	s3	s4	s2	s5	s1
p5	s3	s1	s5	s2	s4

表A3.6：②結果(教員視点)

学生	1位	2位	3位	4位	5位
s1	p3	p5	p1	p4	p2
s2	p1	p2	p3	p4	p5
s3	p1	p4	p3	p5	p2
s4	p2	p5	p4	p1	p3
s5	p4	p1	p5	p2	p3

表A3.7：③結果(学生視点)

教員	1位	2位	3位	4位	5位
p1	s4	s1	s3	s2	s5
p2	s5	s1	s2	s3	s4
p3	s2	s3	s1	s5	s4
p4	s3	s4	s2	s5	s1
p5	s3	s1	s5	s2	s4

表A3.8：③結果(教員視点)

以上のマッチング結果をもとに、DA アルゴリズムの安定性・耐戦略性・申請者最適の 3 つの性質について考える。

・安定性

①ボストン・メカニズム(学生側申請)によるマッチング結果(表 A.3.3、表 A.3.4)によると、(s2, p2)、(s2, p3)、(s2, p4)のペアはそれぞれ自身がマッチしている相手よりも互いに対する選好順序が高いのでブロッキングペアである。よって不安定マッチングである。一方、②DA アルゴリズム(学生側申請)によるマッチング結果(表 A.3.5、表 A.3.6)ではそのようなブロッキングペアは存在しないので、安定マッチングが実現されている。

・耐戦略性

①ボストン・メカニズム(学生側申請)によるマッチング結果(表 A.3.3、表 A.3.4)において、s2 は p2、または p3、もしくは p4 を第 1 希望として嘘の申告をすることで、結果的に導出されたマッチング結果よりも効用が高くなるため、耐戦略性を満たさないことが分かる。一方、②DA アルゴリズム(学生側申請)によるマッチング結果(表 A.3.5、表 A.3.6)では学生全員にとって正直申告が最善となっているため、申請者側の耐戦略性を満たしている。

・申請者側最適性

②DA アルゴリズム(学生側申請)によるマッチング結果(表 A.3.5、表 A.3.6)と③DA アルゴリズム(教員側申請)によるマッチング結果(表 A.3.7、表 A.3.8)を比較すると、まず、②の学生側申請の場合は全ての学生が、選好順序が 1 位から 2 位の教員とマッチしているのに対し、全ての教員は 3 位以下の学生とマッチしている。反対に、③の教員側申請の場合は全ての教員が、選好順序が 1 位から 2 位の学生とマッチしているのに対し、全ての学生は 2 位から 4 位の教員とマッチしている。つまり、ナッシュ均衡として起こりうるマッチングのパターンの中で、②DA アルゴリズム(学生側申請)によるマッチング結果は学生最適かつ教員最悪な安定マッチングであるのに対し、③DA アルゴリズム(教員側申請)によるマッチング結果は教員最適かつ学生最悪な安定マッチングである。以上のことから、DA アルゴリズムは申請者側最適を満たしている。

付録 B　ソースコード

B.1　ボストン・メカニズム

```
"""
Boston Mechanism customized for ICU,s graduate research advisor selection system
Author: Someya Naoki
2019/1/30
This source code is written by Python3.
"""
import numpy as np
import collections
import sys
import random
import time
import copy
import pandas as pd
from collections import Counter


studs = []
labs = []
capa = []
studs_list = {}
labs_list = {}
student_ninki = []
labo_ninki = []
labo_sample_distribution =
[0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,4,4,4,
4,4,4,4,4,4,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,8,8,8,8,8,8,9,9,9,9,10,10,10,10,11,12,13,
13,13,14,15,15,15,15,18,19,21,24,24,27,28,33,36]
```

```
def python_list_add(in1, in2):
    wrk = np.array(in1) + np.array(in2)
    return wrk.tolist()

def merge_dict_add_values(d1, d2):
    return dict(Counter(d1) + Counter(d2))

def remove_values_from_list(the_list, val):
   return [value for value in the_list if value != val]

def create_list(STUDENT_NUM,LABO_NUM,TEIIN,STUDENT_LIST_NUM):

    # 引数の数だけ学生を用意
    students = np.arange(STUDENT_NUM)
    students = students.tolist()
    global studs
    studs = students

    global student_ninki
    student_ninki = []
    for student in students:
        student_ninki.append(student + 1)

    student_ninki = np.sqrt(student_ninki)
    student_ninki_round = []
    for ninki in student_ninki:
        student_ninki_round.append(int(round(ninki)))
    student_ninki = student_ninki_round
```

```
print("\n▼ 学生の人気分布\n", student_ninki)

# 引数の数だけ教員を用意
labos = np.arange(LABO_NUM)
labos = labos.tolist()
global labs
labs = labos

global labo_ninki
labo_ninki = []
global labo_sample_distribution
for key in labos:
    for var in range(labo_sample_distribution[key]):
        labo_ninki.append(key)
print("\n▼ 教員の人気分布\n", labo_ninki)

# 学生の希望順位表を作成
students = {key: None for key in students}
for student in students:
    student_pref = []
    labo_ninki_neo = labo_ninki
    # 学生の選好リストを入力値まで作成
    for var in range(0,STUDENT_LIST_NUM):
        first = random.choice(labo_ninki_neo)
        student_pref.append(first)
        for num in labo_ninki_neo:
            if num == first:
                labo_ninki_neo = remove_values_from_list(labo_ninki_neo, num)
    students[student] = student_pref
```

```
    global studs_list
    studs_list = students
    print("\n▼ 学生の選好順位リスト\n",studs_list)

    global firststuds_list
    firststuds_list = studs_list.copy()

    # 教員の希望順位表を作成
    labos = {key: None for key in labos}
    for labo in labos:
        labo_pref = []
        labo_chozen_count = 0;
        student_ninki_neo = []
        for key in studs_list.keys():
            for value in studs_list[key]:
                if labo == value:
                    labo_chozen_count += 1
                    for var in range(0, student_ninki[key]):
                        student_ninki_neo.append(key)
        for var in range(0,labo_chozen_count):
            labo_choice = random.choice(student_ninki_neo)
            labo_pref.append(labo_choice)
            for num in student_ninki_neo:
                if num == labo_choice:
                    student_ninki_neo  =  remove_values_from_list(student_ninki_neo,
num)
        labos[labo] = labo_pref
```

```
    global labs_list
    labs_list = labos
    print("\n▼ 教員の選好順位リスト\n",labs_list)

    # 教員の選好リストの平均人数を算出
    semi_list_sum = 0;
    i = 0;
    for key in labs_list:
        semi_list_sum += len(labs_list[key])
        i += 1
    global labo_list_average
    labo_list_average = semi_list_sum / i
    print('\n▼ 初回マッチングの教員の選好リストの平均人数\n',labo_list_average)

    global capa
    capa = {key: None for key in labos}
    for item in capa:
        capa[item] = TEIIN

def boston_mechanism(studs, labs, capa, studs_list, labs_list):
    labos_addmitted = [-1 for i in studs_list] # The school each student has been admitted to.
    student_count = [0 for i in studs] # The number of students each school has been admitted.
    unadmitted = [i for i in range(len(studs_list)) if labos_addmitted[i] == -1]
    choice = 0 # The current choice level
    # While there are students left without admitted schools
    admit_count = 1;
    while admit_count > 0:
```

```
        admit_count = 0;
        print("#" * 50)
        print("Choice level: {}".format(choice+1))
        # For each school...
        for labo in range(len(labs)):
            # ...with empty spots left
            if student_count[labo] == capa[labo]:
                continue
            # Consider the students who who rather choose a null school at this
            # choice level
            # if choice < 4:
            null_students = [i for i in unadmitted if len(studs_list[i]) > choice and studs_list[i][choice] == -1]
            # Assign those school to a null school
            for ns in null_students:
                labos_addmitted[ns] = -100
                # print("Student {} gets admitted to a null labo.".format(studs[ns]))
            # print("Students with null choice at choice level {}: {}".format(choice+1, [I[i] for i in null_students]))
            # Consider only the students who listed the current school as their current choice
            choiced_students = [i for i in unadmitted if len(studs_list[i]) > choice and studs_list[i][choice] == labo]
            # print("Students with choice {} at school {}: {}".format(choice+1,
            #       C[sc], [I[i] for i in students]))
            # Assign seats to the school following the priority order of the
            # students at the school until there are no spots left at the school
            # or no students left that put the school as their top priority
            i = 0
```

```
            while not student_count[labo] == capa[labo] and len(labs_list[labo]) > i:
                # Check if the student on the school's priority list are in the
                # group of unadmitted students
                if labs_list[labo][i] in choiced_students:
                    # Assign the student to the school
                    labos_addmitted[labs_list[labo][i]] = labo
                    student_count[labo] += 1
                    print("Student {} admitted to labo {}.".format(studs[labs_list[labo][i]],
labs[labo]))
                    admit_count += 1;
                i += 1
        # Get the new group of unadmitted students
        unadmitted = [i for i in range(len(studs_list)) if labos_addmitted[i] == -1]
        choice += 1
        # for student, school in enumerate(labos_addmitted):
        #       print("{}:   {}.".format(studs[student], labs[school] if school > -1 else "Null
school"))
    return labos_addmitted


def boston_mechanism_add(studs_add, labs_add, capa_add, studs_list_add,
labs_list_add):
    labos_addmitted = [-1 for i in studs_list_add] # The school each student has been
admitted to.
    student_count = [0 for i in labs_add] # The number of students each school has
been admitted.
    unadmitted = [i for i in range(len(studs_list_add)) if labos_addmitted[i] == -1]
    choice = 0 # The current choice level
    # While there are students left without admitted schools
    admit_count = 1;
```

```
    while admit_count > 0:
        admit_count = 0;
        print("#" * 50)
        print("Choice level: {}".format(choice+1))
        # For each school...
        for labo in range(len(labs_add)):
            # ...with empty spots left
            if student_count[labo] == capa_add[labo]:
                continue
            # Consider the students who who rather choose a null school at this
            # choice level
            # if choice < 4:
            null_students = [i for i in unadmitted if len(studs_list_add[i]) > choice and
studs_list_add[i][choice] == -1]
            # Assign those school to a null school
            for ns in null_students:
                labos_addmitted[ns] = -100
                print("Student {} gets admitted to a null labo.".format(studs_add[ns]))
            # print("Students with null choice at choice level {}: {}".format(choice+1, [I[i]
for i in null_students]))
            # Consider only the students who listed the current school as their current
choice
            choiced_students = []
            for i in unadmitted:
                if len(studs_list_add[i]) > choice:
                    if studs_list_add[i][choice] == labo:
                        choiced_students.append(i)
            # choiced_students = [i for i in unadmitted if len(studs_list_add[i]) > choice
and studs_list_add[i][choice] == labo]
```

```
                # print("Students with choice {} at school {}: {}".format(choice+1,
                #       C[sc], [I[i] for i in students]))
                # Assign seats to the school following the priority order of the
                # students at the school until there are no spots left at the school
                # or no students left that put the school as their top priority
                i = 0
                while not student_count[labo] == capa_add[labo] and
len(labs_list_add[labo]) > i:
                    # Check if the student on the school's priority list are in the
                    # group of unadmitted students
                    if labs_list_add[labo][i] in choiced_students:
                        # Assign the student to the school
                        labos_addmitted[labs_list_add[labo][i]] = labo
                        student_count[labo] += 1
                        print("Student {} admitted to labo
{}.".format(studs[labs_list_add[labo][i]], labs_add[labo]))
                        admit_count += 1;
                    i += 1
            # Get the new group of unadmitted students
            unadmitted = [i for i in range(len(studs_list_add)) if labos_addmitted[i] == -1]
            choice += 1
            # for student, school in enumerate(labos_addmitted):
            #       print("{}:   {}.".format(studs[student], labs[school] if school > -1 else "Null
school"))
        return labos_addmitted

if __name__ == '__main__':
        start = time.time()
        trials = []
```

```
STUDENT_NUM = int(sys.argv[1])
LABO_NUM = int(sys.argv[2])
TEIIN = int(sys.argv[3])
STUDENT_LIST_NUM = int(sys.argv[4])
TRIAL_NUM = int(sys.argv[5])
# 計算回数
count = 0;
# 試行回数
loop = 0;
# 評価
sum_stability = [];
sum_strategy = [];
sum_student_rank_rate = {}
sum_student_utility = 0;
sum_labo_utility = 0;
sum_count = 0;
sum_labo_rank_rate_mean = 0;
sum_labo_rank_rate_std = 0;
sum_labo_rank_rate_mean = 0;
sum_labo_list_num = [0 * LABO_NUM];
average_student_rank_list = []
average_labo_rank_list = []
sum_under_student_num_list = []

for var in range(0, TRIAL_NUM):
    count = 1

    print("#" * 50)
    print("ここから%s 回目"%(count))
```

```
        print("#" * 50)

        create_list(STUDENT_NUM,LABO_NUM,TEIIN,STUDENT_LIST_NUM)

        labos_addmitted = boston_mechanism(studs, labs, capa, studs_list, labs_list)
        print("*" * 50)
        std = np.arange(len(studs))
        std = std.tolist()
        std = {key: None for key in std}
        for value in std:
            std[value] = labos_addmitted[value]
        print("\n▼ マッチング結果\n",std)

##################################################################

        #2 回目のマッチングに臨む学生
        nomatch_students = []
        adjust_student_list = []
        adjust_labo_list = []
        for value in std:
            if std[value] == -1:
                nomatch_students.append(value)
        nomatch_students_list = copy.deepcopy(nomatch_students)
        nomatch_students = {key: None for key in nomatch_students}
        studs_add = []
        studs_add = np.arange(len(nomatch_students))
        studs_add = studs_add.tolist()

        # もしマッチできなかった学生がいたら以下を実行
```

```
    if studs_add != []:

        count += 1
        print("#" * 50)
        print("ここから%s 回目"%(count))
        print("#" * 50)

        print("\n▼ マッチできなかった学生\n",nomatch_students)
        print("\n▼ マッチできなかった学生(index)\n",studs_add)

        #2 回目のマッチングに臨む教員
        for key in std:
            value = std[key]
            if value != -1:
                capa[value] -= 1;
        for key in list(capa):
            if capa[key] == 0:
                del capa[key]
        print("\n▼ 定員にまだ空きのある教員 with 定員\n",capa)
        free_labos = []
        for key in capa:
            free_labos.append(key)
        print("\n▼ 定員にまだ空きのある教員\n",free_labos)
        labs_add = []
        labs_add = np.arange(len(capa))
        labs_add = labs_add.tolist()
        print("\n▼ 定員にまだ空きのある教員(index)\n",labs_add)

        # 定員にまだ空きのある教員の残り定員
```

```
            capa_add = []
            for key in capa:
                capa_add.append(capa[key])
            print("\n▼ 定員にまだ空きのある教員の残り定員\n",capa_add)

            # labo_ninki を更新
            for var in labo_ninki:
                if var >= len(labs_add):
                    labo_ninki = remove_values_from_list(labo_ninki, var)

            # 2 回目の学生の選考リストを作る
            studs_list_add = {key: None for key in studs_add}
            for applicant in studs_list_add:
                student_pref = []
                labo_ninki_neo = labo_ninki
                for var in range(0,STUDENT_LIST_NUM):
                    first = random.choice(labo_ninki_neo)
                    student_pref.append(first)
                    for num in labo_ninki_neo:
                        if num == first:
                            labo_ninki_neo                                    =
remove_values_from_list(labo_ninki_neo, num)
                studs_list_add[applicant] = student_pref
            print("\n▼ マッチできなかった学生の新たな選考リスト\n",studs_list_add)

            # 2 回目の教員の選考リストを作る
            labs_list_add = {key: None for key in labs_add}
            for program in labs_list_add:
                labo_pref = []
```

```
                    labo_chozen_count = 0;
                    student_ninki_neo = []
                    for key in studs_list_add.keys():
                        for value in studs_list_add[key]:
                            if program == value:
                                labo_chozen_count += 1
                                for var in range(0, student_ninki[key]):
                                    student_ninki_neo.append(key)
                    for var in range(0,labo_chozen_count):
                        labo_choice = random.choice(student_ninki_neo)
                        labo_pref.append(labo_choice)
                        for num in student_ninki_neo:
                            if num == labo_choice:
                                student_ninki_neo                                    =
remove_values_from_list(student_ninki_neo, num)
                    labs_list_add[program] = labo_pref
                print("\n▼ 今回のマッチングでまだ空きのある教員の新たな選考リスト
\n",labs_list_add)


                labos_list_add = {}
                labo_count = 0;
                for key in labs_list_add:
                    if labs_list_add[key] != []:
                        labos_list_add.update({labo_count:labs_list_add[key]})
                        labo_count += 1;


                labos_addmitted = boston_mechanism_add(studs_add, labs_add, capa_add,
studs_list_add, labs_list_add)
                print("*" * 50)
```

```
std_add = np.arange(len(studs_add))
std_add = std_add.tolist()
std_add = {key: None for key in std_add}
for value in std_add:
    std_add[value] = labos_addmitted[value]
print("\n▼ マッチング結果\n",std_add)
print("*" * 50)

free_labos_index = 0;
for i, v in enumerate(nomatch_students):
    print(std_add[i])
    if std_add[i] == -1:
        print("free_labos_index", free_labos_index)
        free_labos_index = -1
        print("free_labos_index", free_labos_index)
    else:
        free_labos_index = std_add[i]
    nomatch_students[v] = free_labos[free_labos_index]
print("マッチング結果（学生側）: ", nomatch_students)
# 学生の選好順位リスト
adjust_student_list = sorted(nomatch_students.items())
adjust_student_list = dict(adjust_student_list)
for i, v in enumerate(adjust_student_list):
    student_list_index = studs_list_add[i]
    for var in range(len(student_list_index)):
        i = student_list_index[var]
        student_list_index[var] = free_labos[i]
    adjust_student_list[v] = student_list_index
print("index 調整済み学生の選好順序リスト", adjust_student_list)
```

```
        # 教員の選好順位リスト
        adjust_labo_list = {key: [] for key in free_labos}
        for i, v in enumerate(adjust_labo_list):
            labo_list_index = labs_list_add[i]
            for var in range(len(labo_list_index)):
                ll_index = labo_list_index[var]
                ns_count = 0
                for key in nomatch_students:
                    if ns_count == ll_index:
                        labo_list_index[var] = key
                    ns_count += 1
            adjust_labo_list[v] = labo_list_index
        print("index 調整済み教員の選好順序リスト", adjust_labo_list)

    loop += 1
    print("\n")
    print("#" * 50)
    print("%s 周目結果"%(loop))
    print("#" * 50)
    print("\n 全ての学生がマッチするまでの計算回数：%s 回 "%(count))
    trials.append(count)

    # 第 1 回目のマッチング結果(学生側)
    std.update(nomatch_students)
    studentMatchs = std
    print("\n▼ 第%s 回目のマッチング結果(学生側)\n"%(loop),studentMatchs)

    # 最終的な選好順序リスト(学生側)
```

```
for key in adjust_student_list:
    for var in studs_list:
        if key == var:
            studs_list[var].extend(adjust_student_list[key])
print("\n▼ 最終的な選好順序リスト(学生側)\n",studs_list)

programMatchs = np.arange(LABO_NUM)
programMatchs = programMatchs.tolist()
programMatchs = {key: [] for key in programMatchs}
for key in programMatchs:
    for var in studentMatchs:
        if key == studentMatchs[var]:
            programMatchs[key].append(var)
print("\n▼ 第%s 回目のマッチング結果(教員側)\n"%(loop),programMatchs)

# 最終的な選好順序リスト(学生側)
for key in adjust_labo_list:
    for var in labs_list:
        if key == var:
            labs_list[var].extend(adjust_labo_list[key])
print("\n▼ 最終的な選好順序リスト(教員側)\n",labs_list)

# 学生のランク
students_rank = np.arange(STUDENT_NUM)
students_rank = students_rank.tolist()
students_rank = {key: None for key in students_rank}
i = 0;
for s_rank in students_rank:
    i = studentMatchs[s_rank]
```

```
                students_rank[s_rank] = studs_list[s_rank].index(i)
            print("\n▼ 学生のランク\n", students_rank)


            # 教員のランク
            labo_rank = np.arange(LABO_NUM)
            labo_rank = labo_rank.tolist()
            labo_rank = {key: [] for key in labo_rank}
            for l_rank in labo_rank:
                for v in programMatchs[l_rank]:
                    labo_rank[l_rank].append(labs_list[l_rank].index(v))
            print("\n▼ 教員のランク\n", labo_rank)

##################################################################


            print("#" * 50)
            print("評価軸")
            print("#" * 50)

            # (a)安定性
            stability = 0;
            envy = 0;
            for sr_key in students_rank:
                if students_rank[sr_key] >= 1:
                    for i in range (0, students_rank[sr_key]):
                        envy = studs_list[sr_key][i]
                        for var in programMatchs[envy]:
                            if                    labs_list[envy].index(sr_key)                    <
programMatchs[envy].index(var):
                                stability += 1;
```

```
                                    break;
            print("\n(a)安定性：", stability)

            # (b)耐戦略性
            strategy = 0;
            strategy_option = 0;
            swith = 0;
            for sr_key in students_rank:
                if students_rank[sr_key] >= 1:
                    for i in range (0, students_rank[sr_key]):
                        strategy_option = studs_list[sr_key][i]
                        if swith == 1:
                            swith = 0
                        else:
                            for var in programMatchs[strategy_option]:
                                if labs_list[strategy_option].index(sr_key) <
programMatchs[strategy_option].index(var):
                                    strategy += 1;
                                    swith = 1;
                                    break;
            print("\n(b)耐戦略性：", strategy)

            # (c)効率性
            student_max_rank = max(students_rank.values())
            student_rank_rate = np.arange(student_max_rank + 1)
            student_rank_rate = student_rank_rate.tolist()
            student_rank_rate = {key: 0 for key in student_rank_rate}
            for key in student_rank_rate:
                for sr_key in students_rank:
```

```
                    if students_rank[sr_key] == key:
                        student_rank_rate[key] += 1;
        print("\n(c)効率性：")
        print("student_rank_rate: ", student_rank_rate)
        for key in student_rank_rate:
            print("第%s 希望の教員とマッチした学生数：%s （%s）"%(key + 1, student_rank_rate[key], student_rank_rate[key] / STUDENT_NUM * 100))

        print(sum_student_rank_rate)
        sum_student_rank_rate = merge_dict_add_values(sum_student_rank_rate, student_rank_rate)
        print(sum_student_rank_rate)

        average_student_rank = 0
        for key in student_rank_rate:
            rank_num = key + 1
            average_student_rank += rank_num * student_rank_rate[key]
        average_student_rank = average_student_rank/STUDENT_NUM
        print("=>学生の効率性：", average_student_rank)

        average_labo_rank = 0
        for l_rank in labo_rank:
            for value in labo_rank[l_rank]:
                rank_num = value + 1
                average_labo_rank += rank_num
        average_labo_rank = average_labo_rank/LABO_NUM
        print("=>教員の効率性：", average_labo_rank)

        # (d)衡平性
```

```
            kakusa_list = []
            for key in students_rank:
                kakusa_list.append(students_rank[key])
            kakusa_list_nparray = np.array(kakusa_list)
            kakusa_list_nparray += 1;
            kakusa_list_df                                                    =
pd.DataFrame(pd.Series(kakusa_list_nparray.ravel()).describe()).transpose()
            print("\n(d)格差：\n", kakusa_list_df)
            print("・平均順位：", kakusa_list_df.mean()["mean"])
            print("・順位の標準偏差：", kakusa_list_df.mean()["std"])

            # (e)実現可能性
            print("\n(e)実現可能性：")
            print("・完了までの回数：", count)
            labo_list_num = [] #  各教員の選好リスト数
            labo_list_average = 0; #  教員の選好リスト数の平均
            for key in labs_list:
                labo_list_num.append(len(labs_list[key]))
            labo_list_num.sort()
            sum_labo_list_num = python_list_add(sum_labo_list_num, labo_list_num)
            labo_list_num_nparray = np.array(labo_list_num)
            labo_list_num_df                                                  =
pd.DataFrame(pd.Series(labo_list_num_nparray.ravel()).describe()).transpose()
            print("・教員の選好リストの数\n", labo_list_num_df)
            sum_stability.append(stability)
            sum_strategy.append(strategy)
            sum_student_utility += average_student_rank
            average_student_rank_list.append(average_student_rank)
            sum_labo_utility += average_labo_rank
```

```
            average_labo_rank_list.append(average_labo_rank)
            sum_count += count
            sum_labo_rank_rate_mean += kakusa_list_df.mean()["mean"]
            sum_labo_rank_rate_std += kakusa_list_df.mean()["std"]

            sum_under_student_num = 0
            for key in student_rank_rate:
                if key > 1:
                    print(student_rank_rate[key])
                    sum_under_student_num += student_rank_rate[key]
            sum_under_student_num_list.append(sum_under_student_num)
            print(sum_under_student_num_list)

############################################################

        print("#" * 50)
        print("最終結果")
        print("#" * 50)

        print("\n(a)安定性：", sum_stability)
        sum_stability_nparray = np.array(sum_stability)
        sum_stability_df                                                    =
pd.DataFrame(pd.Series(sum_stability_nparray.ravel()).describe()).transpose()
        print(sum_stability_df)
        print("・平均値：", sum_stability_df.mean()["mean"])
        print("・標準偏差：", sum_stability_df.mean()["std"])

        print("\n(b)耐戦略性：", sum_strategy)
        sum_strategy_nparray = np.array(sum_strategy)
```

```
    sum_strategy_df =
pd.DataFrame(pd.Series(sum_strategy_nparray.ravel()).describe()).transpose()
    print(sum_strategy_df)
    print("・平均値：", sum_strategy_df.mean()["mean"])
    print("・標準偏差：", sum_strategy_df.mean()["std"])

    print("\n")
    for key in sum_student_rank_rate:
        print("第%s 希望の教員とマッチした学生数の平均：%s （%s）"%(key + 1,
sum_student_rank_rate[key]/TRIAL_NUM, sum_student_rank_rate[key] /
STUDENT_NUM * 100/TRIAL_NUM))
    print("\n(c)学生の効率性：")
    print(average_student_rank_list)
    print("・平均順位：", sum_labo_rank_rate_mean/TRIAL_NUM)
    print("・順位の標準偏差：", sum_labo_rank_rate_std/TRIAL_NUM)

    print("\n(d)教員の効率性：")
    print(average_labo_rank_list)
    print("・平均順位：", sum_labo_utility/TRIAL_NUM)
    average_labo_rank_list_nparray = np.array(average_labo_rank_list)
    average_labo_rank_list_df =
pd.DataFrame(pd.Series(average_labo_rank_list_nparray.ravel()).describe()).transpose()
    print("・順位の標準偏差：", average_labo_rank_list_df.mean()["std"])

    print("\n(e)学生間の格差：")
    print("・第 3 希望以下の教員に所属した学生数：", sum_under_student_num_list)
    sum_under_student_num_list_nparray = np.array(sum_under_student_num_list)
    sum_under_student_num_list_df =
pd.DataFrame(pd.Series(sum_under_student_num_list_nparray.ravel()).describe()).trans
```

```
pose()
    print(sum_under_student_num_list_df)
    print("・平均値：", sum_under_student_num_list_df.mean()["mean"])
    print("・標準偏差：", sum_under_student_num_list_df.mean()["std"])

    average_labo_list_num = []
    for v in sum_labo_list_num:
        v /= TRIAL_NUM
        average_labo_list_num.append(v)
    print("\n(f)教員の負担：")
    print('・各教員の選好リスト数の平均(昇順)')
    print(average_labo_list_num)
    average_labo_list_num_nparray = np.array(average_labo_list_num)
    average_labo_list_num_df                                                    =
pd.DataFrame(pd.Series(average_labo_list_num_nparray.ravel()).describe()).transpose()
    print(average_labo_list_num_df)

    print("\n(g)実現可能性：")
    print(trials)
    print("・平均計算回数：", sum_count/TRIAL_NUM)
    average = sum_count / loop
    trials_nparray = np.array(trials)
    df = pd.DataFrame(pd.Series(trials_nparray.ravel()).describe()).transpose()
    print(df)

    elapsed_time = time.time() - start
    print("\n・アルゴリズム計算時間の平均")
    print("{0}".format(elapsed_time/TRIAL_NUM) + "[秒]\n")
```

B.2　DA アルゴリズム

```
"""
DA Algorithm customized for ICU,s graduate research advisor selection system
Author: Someya Naoki
2019/1/30
This source code is written by Python3.
"""
import numpy as np
import random
import pandas as pd
import sys
import time
import collections
import copy
from collections import Counter


# 学生の選好リスト
rankApplicant = {}
firstRankApplicant = {}
# ゼミの選好リスト
rankProgram = {}
# 各ゼミの定員
positionProgram = {}
# 最終的な組み合わせ
programMatchs    = {}
# 暫定ゼミを有していない学生の集合
freeApplicant    = []
# checkApplicant   = copy.deepcopy(rankApplicant)
checkApplicant = {}
```

```
# 今回のマッチングでどのゼミともマッチできなかった学生の集合
nomatchApplicant = []
# evaluation_vars
entire_student_pref_list = {}
akiaruProgram = []
simekiriProgram = []
student_ninki = []
labo_ninki = []
labo_sample_distribution =
[0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,4,4,4,
4,4,4,4,4,4,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,8,8,8,8,8,9,9,9,9,10,10,10,10,11,12,13,
13,13,14,15,15,15,15,18,19,21,24,24,27,28,33,36]

def python_list_add(in1, in2):
    wrk = np.array(in1) + np.array(in2)
    return wrk.tolist()

def merge_dict_add_values(d1, d2):
    return dict(Counter(d1) + Counter(d2))

def remove_values_from_list(the_list, val):
   return [value for value in the_list if value != val]

def create_list(STUDENT_NUM,LABO_NUM,TEIIN,STUDENT_LIST_NUM):
    students = np.arange(STUDENT_NUM)
    students = students.tolist()
    print("\n▼ 学生一覧\n",students)

    global student_ninki
```

```
student_ninki = []
for student in students:
    student_ninki.append(student + 1)

student_ninki = np.sqrt(student_ninki)
student_ninki_round = []
for ninki in student_ninki:
    student_ninki_round.append(int(round(ninki)))
student_ninki = student_ninki_round
print("\n▼ 学生の人気分布\n", student_ninki)

labos = np.arange(LABO_NUM)
labos = labos.tolist()
print("\n▼ ゼミ一覧\n",labos)

global labo_ninki
labo_ninki = []
global labo_sample_distribution
for key in labos:
    for var in range(labo_sample_distribution[key]):
        labo_ninki.append(key)
print("\n▼ ゼミの人気分布\n", labo_ninki)

# 学生の希望順位表を作成
students = {key: None for key in students}
for student in students:
    student_pref = []
    labo_ninki_neo = labo_ninki
    for var in range(0,STUDENT_LIST_NUM):
```

```
            first = random.choice(labo_ninki_neo)
            student_pref.append(first)
            for num in labo_ninki_neo:
                if num == first:
                    labo_ninki_neo = remove_values_from_list(labo_ninki_neo, num)
        students[student] = student_pref

    global rankApplicant
    rankApplicant = students
    print("\n▼ 学生の選好順位リスト\n",rankApplicant)

    global firstRankApplicant
    firstRankApplicant = rankApplicant.copy()

    global entire_student_pref_list
    entire_student_pref_list = rankApplicant.copy()

    # ゼミの希望順位表を作成
    labos = {key: None for key in labos}
    for labo in labos:
        labo_pref = []
        labo_chozen_count = 0;
        student_ninki_neo = []
        for key in rankApplicant.keys():
            for value in rankApplicant[key]:
                if labo == value:
                    labo_chozen_count += 1
                    for var in range(0, student_ninki[key]):
                        student_ninki_neo.append(key)
```

```
            for var in range(0,labo_chozen_count):
                labo_choice = random.choice(student_ninki_neo)
                labo_pref.append(labo_choice)
                for num in student_ninki_neo:
                    if num == labo_choice:
                        student_ninki_neo  =  remove_values_from_list(student_ninki_neo,
num)
            labos[labo] = labo_pref


        global rankProgram
        rankProgram = labos
        print("\n▼ ゼミの選好順位リスト\n",rankProgram)


        # ゼミの定員リストを作成
        labos = {key: None for key in labos}
        for labo in labos:
            labos[labo] = TEIIN


        global positionProgram
        positionProgram = labos
        print("\n ゼミの定員リスト\n",positionProgram)


        global checkApplicant
        checkApplicant   = copy.deepcopy(rankApplicant)


def matching(applicant):
    '''Find the free program available to a applicant '''
    # 学生 X のマッチング開始
    print('\n')
```

```
  print("Matching Student %s"%(applicant))
  # 学生 X の選好リストを最新状態にする
  rankApplicant[applicant] = list(checkApplicant[applicant])
  '''If Applicant not have program again, remove from free Applicant '''
  # もし学生 X の選好リストが空の場合
  if(len(rankApplicant[applicant])==0):
    # 暫定ゼミがなくマッチ希望の学生の集合から学生 X を消す
    freeApplicant.remove(applicant)
    nomatchApplicant.append(applicant)
    # 学生 X はどのゼミともマッチしなかった
    print('- Student %s does not have program to check again '%(applicant))
  # もし学生 X の選好リストにゼミがある場合
  else:
    # 学生 X の先行順位表の中のゼミそれぞれに対して以下を実行
    for program in rankApplicant[applicant]:
      #Cek whether program is full or not
      # もしその候補ゼミの定員に空きがあれば
      if len(programMatchs[program]) < positionProgram[program]:
        # もし学生 X の名前が候補ゼミの選考リストに存在しなかったら
        if applicant not in rankProgram[program]:
          # 「学生 X は候補ゼミの選考リストに存在しなかった」と表示
          print('- Student %s does not exist in list applicant in labo %s
'%(applicant,program))
          # 学生 X の選考リストから候補ゼミの名前を消す
          checkApplicant[applicant].remove(program)
        # もし学生 X の名前が候補ゼミの選考リストに存在したら
        else:
          # 学生 X は候補ゼミとマッチする
          programMatchs[program].append(applicant)
```

```
            # 暫定ゼミがなくマッチ希望の学生の集合から学生 X を消す
            freeApplicant.remove(applicant)
            # 「学生 X は暫定ゼミとマッチし、暫定ゼミがなくマッチ希望の学生ではない」と表示
            print('- Student %s is no longer a free applicant and is now tentatively get labo %s'%(applicant, program))
            break
        # もしその候補ゼミの定員に空きがなければ
        else:
          # 「候補ゼミの定員に空きがない」と表示
          print('- labo %s is full (%s participant) '%(program,positionProgram[program]))
          # もし学生 X の名前が候補ゼミの選考リストに存在しなかったら
          if applicant not in rankProgram[program]:
            # 「学生 X は候補ゼミの選考リストに存在しなかった」と表示
            print('- Student %s does not exist in list applicant in labo %s '%(applicant,program))
            # 学生 X の選考リストから候補ゼミの名前を消す
            checkApplicant[applicant].remove(program)
          # もし学生 X の名前が候補ゼミの選考リストに存在したら
          else :
            # get applicant who can remove,
            # 学生 X を applicantRemove とする
            applicantRemove = applicant
            # 候補ゼミの暫定学生それぞれに対して以下を実行
            for applicantMatch in programMatchs[program]:
              # もし学生 X のほうが暫定学生よりも候補ゼミにとって希望順位が高かったら
              if rankProgram[program].index(applicantRemove) < rankProgram[program].index(applicantMatch):
                # 繰り返しにより、最も希望順位の低い学生が不採用学生となる
```

```
                applicantRemove = applicantMatch
            # もし不採用学生がいまだ学生Xだったら
            if applicantRemove==applicant:
                # 「候補ゼミの選考リストにおける学生Xの順位は他の暫定学生よりも低い」と表示
                print('- Rank Applicant %s in labo %s is bigger then other current applicant match '%(applicant,program))
                # 学生Xの選考リストから候補ゼミの名前を消す
                checkApplicant[applicant].remove(program)
            # もし不採用学生が学生Xではなかったら
            else:
                # 「学生Xは不採用学生よりも候補ゼミにとって希望順位が高い」と表示
                print('- Student %s is better than Student %s'%(applicant, applicantRemove))
                # 「不採用学生を暫定学生から外し、新しく学生Xと候補ゼミがマッチした」と表示
                print('- Making Student %s free again.. and tentatively match Student %s and labo %s'%(applicantRemove, applicant, program))

                #The new applicant have match
                # 暫定ゼミがなくマッチ希望の学生の集合から学生Xを消す
                freeApplicant.remove(applicant)
                # 学生Xは候補ゼミとマッチする
                programMatchs[program].append(applicant)

                #The old applicant is now not match anymore
                # 不採用学生を暫定ゼミがなくマッチ希望の学生の集合に加える
                freeApplicant.append(applicantRemove)
                # 不採用学生を暫定学生から外す
                programMatchs[program].remove(applicantRemove)
```

```
                break


if __name__ == '__main__':
    start = time.time()
    trials = []
    STUDENT_NUM = int(sys.argv[1])
    LABO_NUM = int(sys.argv[2])
    TEIIN = int(sys.argv[3])
    STUDENT_LIST_NUM = int(sys.argv[4])
    TRIAL_NUM = int(sys.argv[5])
    # 計算回数
    count = 0;
    # 試行回数
    loop = 0;
    # 評価
    sum_stability = [];
    sum_strategy = [];
    sum_student_rank_rate = {}
    sum_student_utility = 0;
    sum_labo_utility = 0;
    sum_count = 0;
    sum_labo_rank_rate_mean = 0;
    sum_labo_rank_rate_std = 0;
    sum_labo_rank_rate_mean = 0;
    sum_labo_list_num = [0 * LABO_NUM];
    average_student_rank_list = []
    average_labo_rank_list = []
    sum_under_student_num_list = []
```

```
    for var in range(0, TRIAL_NUM):
        count = 1
        print("#" * 50)
        print("ここから%s 回目"%(count))
        print("#" * 50)
        create_list(STUDENT_NUM,LABO_NUM,TEIIN,STUDENT_LIST_NUM)
        # init all applicant free and not have program
        for applicant in rankApplicant.keys():
            freeApplicant.append(applicant)
        # init programMatch
        for program in rankProgram.keys():
            programMatchs[program]=[]
        # Matching algorithm until stable match terminates
        while (len(freeApplicant) > 0):
            for applicant in freeApplicant:
                matching(applicant)
        print("\n▼ 第%s 回目のマッチング結果\n"%(count),programMatchs)
        print("\n▼ 第%s 回目のマッチングでどのゼミともマッチできなかった学生の人数
\n"%count,len(nomatchApplicant))

        while (len(nomatchApplicant) > 0):
            count += 1
            print("#" * 50)
            print("ここから%s 回目"%(count))
            print("#" * 50)
            # まだ所属ゼミが決まっていない学生の集合
            rankApplicant = {key: None for key in nomatchApplicant}
            print("\n▼ 第%s 回目のマッチングの時点でまだ所属ゼミが決まっていない学
```

```
生の集合\n"%count,rankApplicant)
            # 各ゼミの残り定員数
            for program in positionProgram:
                positionProgram[program] = positionProgram[program] -
len(programMatchs[program])
                if positionProgram[program] == 0:
                    simekiriProgram.append(program)
                else:
                    akiaruProgram.append(program)
            # 1回目のマッチを終えてのそれぞれのゼミの残り定員数
            print("\n▼ 各ゼミの残り定員数\n",positionProgram)
            print("\n▼ 募集を締め切ったゼミ\n",simekiriProgram)
            for key in list(positionProgram):
                if positionProgram[key] == 0:
                    del(positionProgram[key])
                    for var in labo_ninki:
                        if var == key:
                            labo_ninki = remove_values_from_list(labo_ninki, var)
            akiaruProgram = positionProgram.keys()
            print("\n▼ 定員にまだ空きのあるゼミ\n",akiaruProgram)
            print("\n▼ 定員にまだ空きのあるゼミの残り定員数\n",positionProgram)
            # positionProgram を初期化
            for key in positionProgram:
                positionProgram[key] = TEIIN
            if len(akiaruProgram) < 3:
                STUDENT_LIST_NUM = len(akiaruProgram)
            # 2回目の学生の選考リストを作る
            for applicant in rankApplicant:
                student_pref = []
```

```
                    labo_ninki_neo = labo_ninki
                    for var in range(0,STUDENT_LIST_NUM):
                        first = random.choice(labo_ninki_neo)
                        student_pref.append(first)
                        for num in labo_ninki_neo:
                            if num == first:
                                labo_ninki_neo                                    =
remove_values_from_list(labo_ninki_neo, num)
                    student_priority = student_pref
                    rankApplicant[applicant] = student_priority
                print("\n▼ マッチできなかった学生の新たな選考リスト\n",rankApplicant)

                for key in rankApplicant:
                    for var in entire_student_pref_list:
                        if key == var:
                            entire_student_pref_list[var].extend(rankApplicant[key])

                # 2回目のゼミの選考リストを作る
                for program in rankProgram:
                    labo_pref = []
                    labo_chozen_count = 0;
                    student_ninki_neo = []
                    for key in rankApplicant.keys():
                        for value in rankApplicant[key]:
                            if program == value:
                                labo_chozen_count += 1
                                for var in range(0, student_ninki[key]):
                                    student_ninki_neo.append(key)
                    for var in range(0,labo_chozen_count):
```

```
                    labo_choice = random.choice(student_ninki_neo)
                    labo_pref.append(labo_choice)
                    for num in student_ninki_neo:
                        if num == labo_choice:
                            student_ninki_neo                                    =
remove_values_from_list(student_ninki_neo, num)
                rankProgram[program].extend(labo_pref)
            print("\n▼ 今回のマッチングでまだ空きのあるゼミの新たな選考リスト
\n",rankProgram)
            checkApplicant   = copy.deepcopy(rankApplicant)
            # init all applicant free and not have program
            for applicant in rankApplicant.keys():
                freeApplicant.append(applicant)

            nomatchApplicant = []
            akiaruProgram = []
            simekiriProgram = []
            labo_ninki = labo_ninki

            # Matching algorithm until stable match terminates
            while (len(freeApplicant) > 0):
                for applicant in freeApplicant:
                    matching(applicant)
            print("\n▼ 第%s 回目のマッチング結果(ゼミ)\n"%(count),programMatchs)
            print("\n▼ 第%s 回目のマッチングでどのゼミともマッチできなかった学生の
人数\n"%count,len(nomatchApplicant))

        loop += 1
        print("\n")
```

```
        print("#" * 50)
        print("%s 周目結果"%(loop))
        print("#" * 50)
        print("\n 全ての学生がマッチするまでの計算回数：%s 回 "%(count))
        trials.append(count)

        student_num_of_each_labo = []
        for key in programMatchs:
            student_num_of_each_labo.append(len(programMatchs[key]))
        print("\n▼ 各ゼミの学生数)\n", student_num_of_each_labo)
        student_num_of_each_labo_nparray = np.array(student_num_of_each_labo)
        student_num_of_each_labo_df =
pd.DataFrame(pd.Series(student_num_of_each_labo_nparray.ravel()).describe()).transpo
se()
        print(student_num_of_each_labo_df)

        print("\n▼ 第%s 回目のマッチング結果(ゼミ側)\n"%(loop),programMatchs)
        print("\n▼ 最終的な選好順序リスト(ゼミ側)\n",rankProgram)
        studentMatchs = np.arange(STUDENT_NUM)
        studentMatchs = studentMatchs.tolist()
        studentMatchs = {key: None for key in studentMatchs}
        for key in studentMatchs:
            for var in programMatchs:
                for value in programMatchs[var]:
                    if key == value:
                        studentMatchs[key] = var
        print("\n▼ 第%s 回目のマッチング結果(学生側)\n"%(loop),studentMatchs)
        print("\n▼ 最終的な選好順序リスト(学生側)\n",entire_student_pref_list)
```

```
# ゼミの人気分布を再度作成
labos_neo = np.arange(LABO_NUM)
labos_neo = labos_neo.tolist()
for key in labos_neo:
    for var in range(labo_sample_distribution[key]):
        labo_ninki.append(key)

# 学生のランク
students_rank = np.arange(STUDENT_NUM)
students_rank = students_rank.tolist()
students_rank = {key: None for key in students_rank}
for s_rank in students_rank:
    for key in programMatchs:
        for value in programMatchs[key]:
            if value == s_rank:
                for frav in firstRankApplicant[s_rank]:
                    if frav == key:
                        rank = firstRankApplicant[value].index(key)
                        students_rank[s_rank] = rank
for s_rank in students_rank:
    if students_rank[s_rank] is None:
        students_rank[s_rank] = 3
print("\n▼ 学生のランク\n", students_rank)

# ゼミのランク
labo_rank = np.arange(LABO_NUM)
labo_rank = labo_rank.tolist()
labo_rank = {key: [] for key in labo_rank}
```

```
        for l_rank in labo_rank:
            for pm_value in programMatchs[l_rank]:
                labo_rank[l_rank].append(rankProgram[l_rank].index(pm_value))
        print("\n▼ ゼミのランク\n", labo_rank)

##################################################################

        print("#" * 50)
        print("%s 周目評価軸"%(loop))
        print("#" * 50)

        # (a)安定性
        stability = 0;
        envy = 0;
        for sr_key in students_rank:
            if students_rank[sr_key] >= 1:
                for i in range (0, students_rank[sr_key]):
                    envy = entire_student_pref_list[sr_key][i]
                    for var in programMatchs[envy]:
                        if rankProgram[envy].index(sr_key) <
programMatchs[envy].index(var):
                            stability += 1;
                            break;
        print("\n(a)安定性：", stability)

        # (b)耐戦略性
        strategy = 0;
        strategy_option = 0;
        swith = 0;
```

```
        for sr_key in students_rank:
            if students_rank[sr_key] >= 1:
                for i in range (0, students_rank[sr_key]):
                    strategy_option = entire_student_pref_list[sr_key][i]
                    if swith == 1:
                        swith = 0;
                    else:
                        for var in programMatchs[strategy_option]:
                            if rankProgram[strategy_option].index(sr_key) < programMatchs[strategy_option].index(var):
                                strategy += 1;
                                swith = 1;
                                break;
        print("\n(b)耐戦略性：", strategy)

        # (c)効率性
        student_max_rank = max(students_rank.values())
        student_rank_rate = np.arange(student_max_rank + 1)
        student_rank_rate = student_rank_rate.tolist()
        student_rank_rate = {key: 0 for key in student_rank_rate}
        for key in student_rank_rate:
            for sr_key in students_rank:
                if students_rank[sr_key] == key:
                    student_rank_rate[key] += 1;
        print("\n(c)効率性：")
        for key in student_rank_rate:
            print("第%s 希望のゼミとマッチした学生数：%s （%s）"%(key + 1, student_rank_rate[key], student_rank_rate[key] / STUDENT_NUM * 100))
        sum_student_rank_rate = merge_dict_add_values(sum_student_rank_rate,
```

```
student_rank_rate)

        average_student_rank = 0
        for key in student_rank_rate:
            rank_num = key + 1
            average_student_rank += rank_num * student_rank_rate[key]
        average_student_rank = average_student_rank/STUDENT_NUM
        print("=>学生の効率性：", average_student_rank)

        average_labo_rank = 0
        for l_rank in labo_rank:
            for value in labo_rank[l_rank]:
                rank_num = value + 1
                average_labo_rank += rank_num
        average_labo_rank = average_labo_rank/LABO_NUM
        print("=>教員の効率性：", average_labo_rank)

        kakusa_list = []
        for key in students_rank:
            kakusa_list.append(students_rank[key])
        kakusa_list_nparray = np.array(kakusa_list)
        kakusa_list_nparray += 1;
        kakusa_list_df                                                    =
pd.DataFrame(pd.Series(kakusa_list_nparray.ravel()).describe()).transpose()
        print("\n(d)格差：\n", kakusa_list_df)
        print("・平均順位：", kakusa_list_df.mean()["mean"])
        print("・順位の標準偏差：", kakusa_list_df.mean()["std"])
        # (g)実現可能性
        print("\n(e)実現可能性：")
```

```
            print("・完了までの回数：", count)
            labo_list_num = [] #  各ゼミの選好リスト数
            labo_list_average = 0; #  ゼミの選好リスト数の平均
            #  ゼミの選好リストの人数を算出
            for key in rankProgram:
                labo_list_num.append(len(rankProgram[key]))
            labo_list_num.sort()
            sum_labo_list_num = python_list_add(sum_labo_list_num, labo_list_num)
            labo_list_num_nparray = np.array(labo_list_num)
            labo_list_num_df                                                    =
pd.DataFrame(pd.Series(labo_list_num_nparray.ravel()).describe()).transpose()
            print("・ゼミの選好リストの数\n", labo_list_num_df)

            sum_stability.append(stability)
            sum_strategy.append(strategy)
            sum_student_utility += average_student_rank
            average_student_rank_list.append(average_student_rank)
            sum_labo_utility += average_labo_rank
            average_labo_rank_list.append(average_labo_rank)
            sum_count += count
            sum_labo_rank_rate_mean += kakusa_list_df.mean()["mean"]
            sum_labo_rank_rate_std += kakusa_list_df.mean()["std"]

            sum_under_student_num = 0
            for key in student_rank_rate:
                if key > 1:
                    print(student_rank_rate[key])
                    sum_under_student_num += student_rank_rate[key]
            sum_under_student_num_list.append(sum_under_student_num)
```

```
            print(sum_under_student_num_list)

##################################################################

        print("#" * 50)
        print("最終結果")
        print("#" * 50)

        print("\n(a)安定性：", sum_stability)
        sum_stability_nparray = np.array(sum_stability)
        sum_stability_df                                                    =
pd.DataFrame(pd.Series(sum_stability_nparray.ravel()).describe()).transpose()
        print(sum_stability_df)
        print("・平均値：", sum_stability_df.mean()["mean"])
        print("・標準偏差：", sum_stability_df.mean()["std"])

        print("\n(b)耐戦略性：", sum_strategy)
        sum_strategy_nparray = np.array(sum_strategy)
        sum_strategy_df                                                     =
pd.DataFrame(pd.Series(sum_strategy_nparray.ravel()).describe()).transpose()
        print(sum_strategy_df)
        print("・平均値：", sum_strategy_df.mean()["mean"])
        print("・標準偏差：", sum_strategy_df.mean()["std"])

        print("\n")
        for key in sum_student_rank_rate:
                print("第%s 希望のゼミとマッチした学生数の平均：%s （%s）"%(key + 1,
sum_student_rank_rate[key]/TRIAL_NUM,          sum_student_rank_rate[key]          /
STUDENT_NUM * 100/TRIAL_NUM))
```

```
        print("\n(c)学生の効率性：")
        print(average_student_rank_list)
        print("・平均順位：", sum_labo_rank_rate_mean/TRIAL_NUM)
        print("・順位の標準偏差：", sum_labo_rank_rate_std/TRIAL_NUM)

        print("\n(d)教員の効率性：")
        print(average_labo_rank_list)
        print("・平均順位：", sum_labo_utility/TRIAL_NUM)
        average_labo_rank_list_nparray = np.array(average_labo_rank_list)
        average_labo_rank_list_df                                                    =
pd.DataFrame(pd.Series(average_labo_rank_list_nparray.ravel()).describe()).transpose()
        print("・順位の標準偏差：", average_labo_rank_list_df.mean()["std"])

        print("\n(e)学生間の格差：")
        print("・第 3 希望以下のゼミに所属した学生数：", sum_under_student_num_list)
        sum_under_student_num_list_nparray = np.array(sum_under_student_num_list)
        sum_under_student_num_list_df                                                =
pd.DataFrame(pd.Series(sum_under_student_num_list_nparray.ravel()).describe()).trans
pose()
        print(sum_under_student_num_list_df)
        print("・平均値：", sum_under_student_num_list_df.mean()["mean"])
        print("・標準偏差：", sum_under_student_num_list_df.mean()["std"])

        average_labo_list_num = []
        for v in sum_labo_list_num:
            v /= TRIAL_NUM
            average_labo_list_num.append(v)
        print("\n(f)教員の負担：")
        print('・各ゼミの選好リスト数の平均(昇順)')
```

```
    print(average_labo_list_num)
    average_labo_list_num_nparray = np.array(average_labo_list_num)
    average_labo_list_num_df =
pd.DataFrame(pd.Series(average_labo_list_num_nparray.ravel()).describe()).transpose()
    print(average_labo_list_num_df)

    print("\n(g)実現可能性：")
    print(trials)
    print("・平均計算回数：", sum_count/TRIAL_NUM)
    average = sum_count / loop
    trials_nparray = np.array(trials)
    # print(trials_nparray)
    df = pd.DataFrame(pd.Series(trials_nparray.ravel()).describe()).transpose()
    print(df)

    elapsed_time = time.time() - start
    print("\n・アルゴリズム計算時間の平均")
    print("{0}".format(elapsed_time/TRIAL_NUM) + "[秒]\n")
```

English Abstract

In this paper, we consider the practical feasibility of Deferred Acceptance Algorithm in one - to - many matching of students and faculty, which is a university 's thesis advisor assignment system. Matching phenomena are often seen in our everyday life. With whom and what to match is an important issue affecting the success of the project and life satisfaction. The optimal pair creation method in such a matching phenomenon has been developed in matching theory which is the application field of game theory, but it has not been fully utilized yet in the real world. For example, the Boston mechanism that is vulnerable to stability and strategy-proof is adopted as the current thesis advisor assignment system of International Christian University. Under such a system, there is a possibility that the student has legitimate dissatisfaction with the matching result or suffers opportunity loss by strategic operation of the preference order. In this paper, therefore, by applying institutional design utilizing the Deferred Acceptance Algorithm at International Christian University's thesis advisor assignment system,

we will consider constructing a mechanism that can realize that students can declare their own honestly and match with the most desirable faculty member among the feasible. First of all, we will provide a precise institutional design in line with the specific concrete situation in the selection system of international Christian University graduation research advisor. Then, we prepare both programs corresponding to the design, and carry out the simulation experiment on the computer. Furthermore, based on the results of the simulation experiments, a comparative analysis of the system is carried out on seven evaluation axes: stability, strategy-proof, the utility of students, the utility of teachers, the disparity between students, faculty burden, and feasibility. Through these, we examine the practical feasibility of Deferred Acceptance Algorithm in matching between students and faculty.