信州大学工学部

学士論文

同順位リストを用いた安定性に基づく研究室配属問題

指導教員　西新 幹彦 准教授

学科　電子情報システム工学科
学籍番号　16T2125G
氏名　那須野 亨

2020 年 2 月 28 日

目次

1　はじめに

大学 4 年次生にとって研究室配属は，その後の卒業研究に対するモチベーションを大きく左右する非常に重要なイベントである．全ての学生が希望の研究室に所属されれば良いが，研究室には定員が設けられている場合がほとんどで，希望上位の研究室に入れない学生も当然出てくる．そのため，各学生の希望にできる限り沿いつつ，全体として不満の少ない（皆が納得できる）配属方法を採用することが求められる．また，研究室配属を公平かつ円滑に進めるためにも，その方法は学生と教員の両者にとって分かりやすく，負担の少ないものであることが望ましい．これらの点でより良い研究室配属の方法を提案することが本研究の目的である．

本研究では，研究室配属の一つの方法として，学生が研究室を好みの順に並べた希望リストを提出し，それを基に学生の成績順で配属を行うものを取り上げる．これと似通った問題として，男女のマッチングを題材とした「安定結婚問題」が広く知られている．これは「安定性」の概念を重要視した手法の一つで，希望リストの下で崩れることのない “安定な” マッチングを求めるものである．

安定結婚問題は男女のペアを題材としたものであるが，これを学生と研究室に置き換えることで，そのまま研究室配属に応用することができる．この配属方法から得られる配属結果は “安定な” もので，学生の希望リストと成績順という二つの基準の下で崩れる事のないバランスのとれたものになる．しかし，安定結婚問題で用いられる希望リストは，明確に順位付けされたもので，かつペアとなり得る相手全員が希望リスト上に存在しなければならない．そのため，選択肢となる研究室が数多くあった場合，学生は明確な好みの差がない研究室までも無理に順位付けしなければならなくなり，希望リストの作成が負担となる．また，この時ある学生によっていい加減に順位付けされた研究室が，他の学生の配属先に悪影響を及ぼす可能性が考えられる．そこで本研究では，この安定結婚問題を基にした研究室配属方法に「同順位リスト」を導入した新たな研究室配属アルゴリズムを提案する．

このアルゴリズムは，学生の希望リスト作成の手間を軽減すると共に，その配属結果がより多くの学生の希望が叶えるものとなることが期待される．

2　安定結婚問題から研究室配属問題へ

ここではまず，安定結婚問題における安定性の概念を詳しく説明し，研究室配属と安定結婚問題との対応関係を示す．また，既存の研究と本研究との関係について述べる．

2.1 安定結婚問題

安定マッチング問題の一種として，男女のペアを考える「安定結婚問題」がある．複数の男女が各人の好みで異性を順序付けした希望リストを持ち，その希望リストに基づいて「安定なマッチング」を求めるのが，この安定結婚問題である [1]．ここで，「マッチング」とは男女のペア n 組からなる集合である．“安定な” の正確な意味は「安定性」の概念と共に後述する．まず，安定結婚問題の例を図 1 に示す．

図 1 では，1〜3 を男性，a〜c を女性とし，各人の好みを希望リストとして表現している．例えば，男性 1 の希望順位 1 位は女性 a である．希望リスト上で○で囲んだものは，この希望リストに対するマッチングの一例である．今，このマッチングを M とすると，マッチング $M = \{(1, a), (2, c), (3, b)\}$ と表せる．マッチング M で男性 1 とペアになっているのが女性 a であることを $M(1) = a$ と表し，同様に女性 a とペアになっているのが男性 1 であることを $M(a) = 1$ と表す．

あるマッチングにおいて，ペアになっていない男女の組が，そのマッチングにおける自身のペアよりもお互いを好むとき，その男女の組を「不安定ペア」とし，そうしたペアの存在しないマッチングを「安定なマッチング」と呼ぶ．不安定ペアとはすなわち，不倫を起こす関係にある男女の組である．図 1 の例 M では，$(3, c)$ が不安定ペアである．よって，M は安定なマッチングではない．

1:	ⓐ	b	c	a:	①	2	3
2:	a	ⓒ	b	b:	1	2	③
3:	c	ⓑ	a	c:	3	②	1

図 1　安定結婚問題の例

どのような希望リストに対しても安定なマッチングは必ず存在し，$\mathcal{O}(n^2)$ 時間でこれを見つけるアルゴリズム「Gale-Shapley アルゴリズム」が知られている [3]．ここでは，簡単にこのアルゴリズムの動作を説明する．アルゴリズムの実行中，男性と女性は「婚約中」もしくは「独身」の二つの状態をとる．最初は全員が独身である．独身の男性は，自分の希望リストの上位の女性から順にプロポーズをしていく．どの男性からプロポーズを始めてもよい．プロポーズを受けた女性は，自分が独身であればそのプロポーズを受け入れ婚約中となり，逆に自分が婚約中であれば，現在既に婚約中の男性とプロポーズしてきた男性とを比べ，自分の好きな方の男性と婚約する．ここで好きな方の男性とは，希望リストでより上位の男性である．振られた男性は再び独身となり，自分の希望リストからその女性を削除する．同じ女性に二度プロポーズすることはない．これを独身の男性がいなくなるまで続ける．図 2 に，このアルゴ

リズムを図 1 の希望リストに対して実行した場合の結果を示す．得られたマッチング M' では，ある男性 m に $M'(m)$ よりも好む女性 w がいたとしても，w は m よりも好む男性 $M'(w)$ とペアになっているため，m は w とペアになれず不倫は成立しない．そういった意味でこのマッチングは安定である．

1:	ⓐ	b	c		a:	①	2	3
2:	a	c	ⓑ		b:	1	②	3
3:	ⓒ	b	a		c:	③	2	1

図 2　Gale-Shapley アルゴリズムの実行結果

2.2　研究室配属問題

安定結婚問題は，一対一のマッチングを対象としたものであるが，こうした安定マッチングの考え方は，大学での研究室配属や，病院への研修医配属など，一対多のマッチングにも応用されている例がある [2]．研究室配属を行う場合であれば，安定結婚問題における男性と女性を，学生と研究室に置き換えて考える．ここでは簡単化のため，研究室の定員を全て一人とし，研究室配属を一対一のマッチングとして話を進める．

安定結婚問題をベースとした研究室配属を行うにあたっては，学生側の希望リストとして，選択肢となる研究室の数と同じサイズの明確に順位付けされた希望リストを用意する必要がある．安定結婚問題で全ての男女がペアになっていたように，研究室配属においても，一度の配属操作で全学生を必ずいずれかの研究室に配属したいとすれば，全研究室が各学生の希望リスト上になければならない．しかし研究室の数が増えれば増えるほど，学生は明確な好みの差がない研究室までも無理に順位付けしなければならなくなり，希望リストの作成に労力がかかる．一方，研究室側の希望リストも学生の人数と同じサイズのものが必要になるが，本研究では学生の成績順で配属を行うことを前提とするため，各研究室が個別に希望リストを作成する必要はない．

また，成績上位の学生が明確な好みの差がない研究室をいい加減に順位付けしてしまうことは，成績下位の学生の配属先に悪影響を及ぼす可能性がある．例えば，ある学生 s_1 が，明確な好みの差がない研究室 r_1 と r_2 をこの順番でなんとなく順位付けし，その結果 r_1 に配属されたとする．この時，s_1 より成績下位の学生 s_2 が r_1 を好んでいたとしても，成績で劣るため s_2 は r_1 に配属されない．もしも s_1 が r_1 と r_2 の順番を変えていたら，s_1 と s_2 の両者にとってより良い配属結果が得られただろう．

そこで本研究では，学生側の希望リストで複数の研究室を一つの順位にまとめ，第 1 希望群，第 2 希望群，第 3 希望群・・・とする「同順位リスト」を用いた研究室配属方法を提案す

る．希望リストに同順位を認めることで，学生は全ての研究室を無理に順位付けする必要がなくなり，リスト作成の負担が軽減される．またこの配属方法では，先の例のように学生同士で希望の研究室が重なった際に，各学生を同順位の他の研究室にうまく移動させることで，全体として配属先研究室の希望順位を上げる効果を同時に得ることが期待される．

2.3 既存の研究との関係

前節で述べた同順位リストを用いたマッチングは，安定結婚問題の拡張の一例として既に知られている [1][4]．既存の研究で扱われている問題は，男女双方の希望リストに同順位を認め，超安定・強安定・弱安定の 3 種類の安定性の基準を設けたもので，それぞれに対応するアルゴリズムが提案されている．安定性の基準が複数あるのは，希望リストに同順位を認めることによって，不倫を起こすペアが互いを好む程度に幅が生じるためである．ここでは，簡単にこの 3 種類の安定性の基準について説明する．

ある人 m がマッチング M の相手として $M(m)$ よりも別の人 a を明確に好むとき $M(m) < a$ と表し，$M(m)$ 以上に a を好むとき $M(m) \leq a$ と表す．$M(m) \leq a$ であるときは，m の希望リスト上で $M(m)$ と a が同順位となっていても良く，$M(m) < a$ であるときは，$M(m)$ と a が明確に順位付けされていなければならない．

あるマッチング M_w が弱安定であるとは，M_w でペアになっていない組 (a, b) で，「$M_w(a) < b$ かつ $M_w(b) < a$」となる組が存在しないことである．これは簡単に言い換えれば，(a, b) の両者が明確にお互いを好んでいなければ不倫は成立しないということである．次に，あるマッチング M_s が強安定であるとは，M_s でペアになっていない組 (a, b) で，「$M_s(a) \leq b$ かつ $M_s(b) < a$」もしくは「$M_s(a) < b$ かつ $M_s(b) \leq a$」となる組が存在しないことである．これは簡単に言い換えれば，(a, b) の両者が互いに現在のペア以上の存在で，かつどちらかが明確に相手を好んでいなければ不倫は成立しないということである．最後に，あるマッチング M_{ss} が超安定であるとは，M_{ss} でペアになっていない組 (a, b) で，「$M_{ss}(a) \leq b$ かつ $M_{ss}(b) \leq a$」となる組が存在しないことである．これはすなわち，(a, b) の両者が互いに現在のペアと同順位の存在であっても不倫するとみなされるということである．

ここまで，簡単に男女の双方が同順位リストを持つ場合の安定性の定義について述べてきたが，本研究では学生の希望リストにのみ同順位を認めたマッチングを扱うため，これら 3 種類のものとは別に不安定ペアを新たに定義する．その後，この不安定ペアの存在しないアルゴリズムを示す．

また，本研究で示すアルゴリズムは，研究室側の希望リストが学生の成績順で全て同一のものであることを前提としているため，従来の研究におけるアルゴリズムに比べ処理の簡単なものになっている．

3　従来の配属方法と同順位リストの導入

ここではまず，学生の成績順で配属を行う安定結婚問題をベースとした研究室配属の方法を「従来の配属方法」として定義し，そこに同順位リストを導入する．

3.1　従来の配属方法

安定結婚問題における男性と女性を学生と研究室に置き換え，従来の配属方法を定義する．繰り返しになるが，ここでは研究室配属を一対一のマッチングとして考える．学生側の希望リストは，安定結婚問題と同様に全ての研究室を明確に順位付けした通常の希望リストを用いる．

図 3 に，1〜5 を学生，A〜E を研究室とし，この配属方法の例を示す．学生はあらかじめ成績順にソートされ，リストに並べられているとする．研究室側の希望リストは全てこの順序になっている．当然 Gale-Shapley アルゴリズムをこの希望リストに適用すれば一対一の安定なマッチングを得られるが，研究室側の希望リストは全て同様であるため，単純に，学生 1 から成績順を守って研究室に割り当てていけば，Gale-Shapley アルゴリズムと同じ操作をしていることになる．具体的には，各学生の第 1 希望の研究室から順に，対象の研究室が空いていればそこへ配属し，空いていなければ希望順位を下げながら空いている研究室を探していく．最終的には，全学生がいずれかの研究室に必ず配属される．

図 3 の希望リスト上に○で囲んでいるものは，サイズ 5 の希望リストに対してこの配属方法を適用した結果である．これは，安定結婚問題における “安定” と同様の意味で，安定な配属結果である．

1:	Ⓑ	C	D	A	E		A:	1	2	3	④	5
2:	Ⓓ	A	E	B	C		B:	①	2	3	4	5
3:	Ⓔ	B	A	D	C		C	1	2	3	4	⑤
4:	B	Ⓐ	E	D	C		D:	1	②	3	4	5
5:	E	A	B	D	Ⓒ		E:	1	2	③	4	5

図 3　従来の配属方法の例

3.2　同順位リストを用いた配属方法

前節で述べた従来の配属方法に，同順位リストを導入する．同順位リストでは，学生が複数の研究室を一つの順位にまとめることを許す．いくつの研究室を同順位とするかも，各人の自

由である．同順位リストを用いた配属でも従来の配属方法と同様に，安定性の概念を重要視する．図 4 に同順位リストの例を示す．同順位となる研究室を “( )”でまとめて表現している．例えば，学生 1 の第 1 希望群の研究室は B,C,D の三つ，第 2 希望群の研究室は A,E の二つである．

1:	(B	C	D)	(A	E)	A:	1	2	3	4	5
2:	(D	A)	E	(B	C)	B:	1	2	3	4	5
3:	(E	B)	(A	D	C)	C	1	2	3	4	5
4:	B	(A	E)	D	C	D:	1	2	3	4	5
5:	(E	A	B)	(D	C)	E:	1	2	3	4	5

図 4　同順位リストの例

同順位リストの導入により，学生は同程度の好みの研究室を無理に順位付けする必要がなくなるため，余計なことに頭を悩ませなくて済む．また，いくつの研究室を同順位としてまとめても良いため，例えば，明確な好みの差がある研究室のみ順位付けをし，それ以外の研究室は全て一つの希望群にまとめてしまうというように，より個人の好みに沿った希望リストを作成することができるようになる．

次に，この同順位リストから学生をどう研究室に配属するかを考えるが，ひとまず同順位を表す “( )”を無視し，従来の配属方法を用いて成績上位の学生から順に研究室に割り当ててみる．図 5 はこの作業を途中まで行い，学生 3 の配属が済んだところまでの状態を示したものである．

1:	(Ⓑ	C	D)	(A	E)	A:	1	2	3	4	5
2:	(Ⓓ	A)	E	(B	C)	B:	①	2	3	4	5
3:	(Ⓔ	B)	(A	D	C)	C	1	2	3	4	5
4:	B	(A	E)	D	C	D:	1	②	3	4	5
5:	(E	A	B)	(D	C)	E:	1	2	③	4	5

図 5　同順位リストの例 2

図 5 で，次は学生 4 の配属先を探すが，学生 4 の第 1 希望の研究室 B には既に学生 1 が配属されている．学生 1 は研究室 B のほかに研究室 C と研究室 D を第 1 希望群としているため，学生 1 にまだ誰も配属されていない研究室 C に移動してもらえば，学生 4 の第 1 希望が叶えられる．同順位リストでは，同じ希望群の研究室はどれも完全に同程度の好みと考えるため，こうした学生の研究室間での入れ替え操作が可能になる．これを以降「玉突き操作」と呼ぶ．

今取り上げた図 5 の例は非常に簡単なものであり，この玉突き操作がより良い配属結果をも

たらすことは明らかである．しかし，より希望リストのサイズが大きな場合には，何人もの学生が関わる複雑な玉突き操作が起こり得るため，どの学生をどう移動させれば全体として一番良い配属結果になるのか一目見て判断することは難しくなる．また，そうした玉突き操作の中で，成績下位の学生の希望を叶えるために，成績上位の学生の配属先研究室の希望順位が下がるようなことがあっては，従来の配属方法のようには配属結果の安定性が保たれなくなってしまうことも考えられる．そこで次章では，こうした玉突き操作を考慮した上で，同順位リストを用いた研究室配属における不安定ペアを新たに定義する．その上で，この不安定ペアの存在しない配属を行うアルゴリズムを示す．

4　同順位アルゴリズム

4.1　不安定ペアの定義

ある研究室配属 M で，学生 s と研究室 r がペアになってるとき，$M(s) = r$ および $M(r) = s$ と書く．s が $M(s)$ よりも研究室 r' を明確に好んでいたとき，次の条件 1 または条件 2 のどちらかに当てはまる場合，(s, r') を不安定ペアとする．

1. s は $M(r')$ よりも成績上位
2. $M(r')$ の希望リストに r' と同順位の別の研究室があるとき，2-1 もしくは 2-2
   - 2-1. 玉突き操作に関わる学生全員の希望順位を変えることなく s を r' に配属できる
   - 2-2. 玉突き操作の際に，s より成績下位の学生の希望順位を下げることで s を r' に配属できる

研究室側の希望リストは全て学生の成績順であるため，(s, r') が条件 1 に当てはまった場合にこれを不安定ペアとするのは安定結婚問題と同様の考え方である．一方の条件 2 では，学生の玉突き操作に関する条件を示している．学生 $M(r')$ の希望リストに r' と同順位の別の研究室があった場合，学生 s を玉突き操作によって研究室 r' に配属できるかどうかを考える．その際，玉突き操作に関わる学生全員が同順位の研究室内での移動か，もしくは s より成績下位の学生の希望順位のみ下げることで入れ替えられる場合には，学生 s を研究室 r' に配属できるとする．こうした玉突き操作を行えば (s, r') をペアにすることが可能であるにも関わらずそうなっていない場合，これを不安定ペアとする．これら二つの条件に当てはまらない場合には，学生 s を研究室 r' に配属するためには必ず s よりも成績上位の学生の希望順位を下げる必要が生じるため，s には自身の成績順位では研究室 $M(s)$ が最良なのだと納得してもらうことにする．そうすることで，マッチングの安定性が保たれると考える．本研究ではこれを，同順位リストを用いた研究室配属における不安定ペアの定義とする．

次節では，こうした不安定ペアの定義を踏まえて，必ず安定な配属を行う「同順位アルゴリ

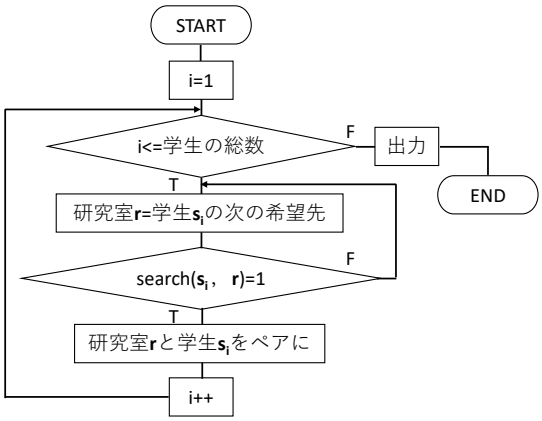

図 6　同順位アルゴリズム 1

ズム」の概要を示す．

4.2　アルゴリズムの概要

同順位アルゴリズムはその実行中，学生と研究室の組を引数として受取り，その学生と研究室をペアにできるかどうかを判定する手続き search を再帰的に呼び出していく．図 6 にこのアルゴリズムの大枠をフローチャートで示す．

アルゴリズムの開始時，学生は予め成績順にソートされているものとする．手続き search は，渡された組をペアにできれば 1 を，できなければ 0 を返す．各学生の希望先研究室に対し，手続き search が 0 を返す限りは，希望順位を下げながら繰り返しこれを呼び出していく．手続き search が 1 を返せば，配属先が “一旦”決まり，成績順で次の学生の配属へと移っていく．ここで，配属先が “一旦”決まると述べたのは，その研究室と同順位の研究室が他にあれば，まだ玉突き操作によって移動する可能性があるためである．成績上位者から順に配属を行っていき，成績最下位の学生の配属先が決まれば，アルゴリズムは終了である．図 7 は手続き search の具体的な処理内容である．

「学生をロック」の詳しい意味は図 8 の例を用いて後述する．この手続き search は，渡された学生と研究室の組を玉突き操作によってペアにできる可能性があると判断すれば，その中で自分自身を再び呼び出す．再帰呼び出しの先で手続き search が 1 を返すまでは，あくまでペアにできるかどうかの判定を行うのみで，実際に学生と研究室はペアにならない．図 9 は図 8 の希望リストに対する手続き search の実際の処理の流れである．以下では，この図 8 と図 9 を用いて，処理の内容を詳しく説明する．

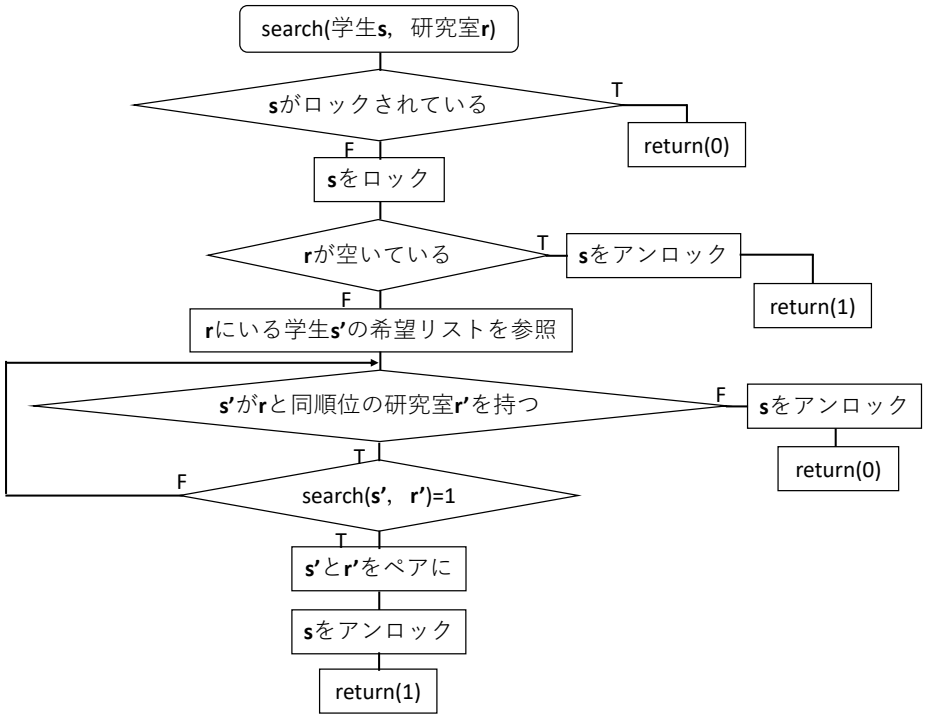

図 7　同順位アルゴリズム 2

1:	(A	B)	(C	D	E)	A:
2:	(B	A)	C	(E	D)	B:
3:	A	C	(B	E	D)	C:
4:	(D	E)	A	(B	C)	D:
5:	D	(A	C)	(E	D)	E:

図 8　説明の為の例

学生 1 と学生 2 に関しては，いずれも第 1 希望の研究室が空いているため，手続き search は 1 を返し配属先が一旦決まる（図 9 の 1,2）.

次に学生 3 は研究室 A を第 1 希望としているが，研究室 A には既に学生 1 が配属されているため，同順位アルゴリズムは学生 1 の希望リストを確認する．すると，学生 1 は研究室 B を同順位で第 1 希望としているため，再帰的に search$(1, B)$ を呼び出す（図 9 の 4）．しかし，研究室 B にも既に学生 2 が配属されているため，今度は学生 2 の希望リストを確認する．学

1. search$(1, A) = 1$
2. search$(2, B) = 1$
3. search$(3, A)$
    4. search$(1, B)$
        5. search$(2, A)$
            6. search$(1, B) = 0$...locked
        7. search$(2, A) = 0$
    8. search$(1, B) = 0$
9. search$(3, A) = 0$
10. search$(3, C) = 1$
11. search$(4, D) = 1$
12. search$(5, D)$
    13. search$(4, E) = 1$
14. search$(5, D) = 1$

図 9　再帰処理の様子

生 2 は研究室 A を同順位で第 1 希望としているため，search$(2, A)$ を呼び出す（図 9 の 5）．同順位アルゴリズムは，ここで再び研究室 A に配属されている学生 1 の希望リストを確認し，search$(1, B)$ を呼び出す（図 9 の 6）．つまり，玉突き操作の結果，自分が移動して学生 3 のために空けようとした研究室に全く別の学生がきてしまう．その結果，処理がループに入る．玉突き操作を行う上では，こうしたケースが多々生じる．そこで手続き search は，最初に必ず引数として渡された学生をロックし，こうしたループを防いでいる．ロックのかかった学生 1 が再び手続き search に渡された段階（図 9 の 6）でこの玉突き操作は成功しないと判断され，手続き search は 0 を返す（図 9 の 6,7,8,9）．学生 1 にはこれ以上別の第 1 希望の研究室がないため，学生 3 は第 2 希望の研究室 C へ配属される（図 9 の 10）．

学生 4 は第 1 希望の研究室 D に一旦配属される（図 9 の 11）．学生 5 は，search$(4, E)$ の呼び出しの結果，学生 4 が空いている研究室 E に移動できると判断されるため（図 9 の 13），第 1 希望の研究室 D に配属される（図 9 の 14）．このように，手続き search を再帰的に呼び出した先で，まだ誰も配属されていない研究室にたどり着ければ，玉突き操作は成功する．図 8 の希望リストの場合には，こうして全学生の配属が終了する．図 10 にこの結果を示す．

1:	(Ⓐ	B)	(C	D	E)	A:	1
2:	(Ⓑ	A)	C	(E	D)	B:	2
3:	A	Ⓒ	(B	E	D)	C:	3
4:	(D	Ⓔ)	A	(B	C)	D:	5
5:	Ⓓ	(A	C)	(E	D)	E:	4

図 10　図 8 の例の結果

4.3　安定性の証明

前節の同順位アルゴリズムが必ず安定な配属，つまり不安定ペアの存在しない配属を行うことを示す．同順位アルゴリズムで行ったある研究室配属 M において，不安定ペア (s, r) が存在すると仮定する．すると，研究室 r は M で $M(r) \neq s$ とペアになっていることになる (この $M(r)$ は本来ここにいてはいけない学生である)．学生 s が M で研究室 r とペアになれなかったということは，同順位アルゴリズムが学生 s の配属先研究室を探す段階で，研究室 r には既に別の学生 s' が配属されていたはずである．この学生 s' は研究室 r から玉突き操作によって移動することはできない．移動できるとすると学生 s が研究室 r とペアになる．つまり，学生 s' と $M(r)$ は全く別の学生ということになる．これは，研究室と学生のペアが一対一であることに矛盾する．よって，同順位アルゴリズムは必ず安定な配属を行う．

5　アルゴリズムの評価

前章までの内容で，同順位アルゴリズムが，従来の配属方法と同様に必ず安定な配属を行うことを保証しつつ，学生の希望リスト作成の負担を軽減できることを示してきた．

2.2 節で，同順位リストを用いた研究室配属では，学生同士で希望の研究室が重なった際に各学生を同順位の他の研究室にうまく移動させることで，全体として配属策研究室の希望順位を上げる効果を得ることが期待されると述べたが，この章では，従来の配属方法による配属結果と同順位アルゴリズムによる配属結果とを比較して，実際にこの効果がどの程度のものであるかを検証する．

5.1　研究室配属シミュレーション

従来の配属方法による配属結果と同順位アルゴリズムによる配属結果とを比較するにあたり，ランダムに生成された希望リストを用いて「研究室配属シミュレーション」を行う．ここでは，この希望リストの具体的な生成方法を説明する．

まず，図 11 左に示したような通常の希望リストを，各研究室の人気の偏りを表現できるよう重みづけした上で，ある一連の研究室の並び

$$\boldsymbol{x} = x_1 x_2 \cdots x_n$$

となる確率が

$$\begin{aligned} P(\boldsymbol{x}) &= \frac{w_{x_1}}{\sum_{k=1}^{n} w_{x_k}} \frac{w_{x_2}}{\sum_{k=1}^{n} w_{x_k} - w_{x_1}} \frac{w_{x_3}}{\sum_{k=1}^{n} w_{x_k} - w_{x_1} - w_{x_2}} \cdots \frac{w_{x_{n-1}}}{w_{x_{n-1}} + w_{x_n}} \frac{w_{x_n}}{w_{x_n}} \\ &= \prod_{j=1}^{n} \frac{w_{x_j}}{\sum_{k=j}^{n} w_{x_k}} \end{aligned}$$

であるように生成する．ここで，w_l は研究室 l の重み，n は研究室の総数である．大きな重みを与えられた研究室ほど，人気があるということを示している．

次に，図 11 右のような同順位リストを，図 11 左の通常の希望リストに，隣同士の研究室が同順位となる確率を $\frac{1}{2}$ として同順位の “( )” を追加し用意する．

1:	B	C	D	A	E		1:	(B	C	D)	(A	E)
2:	D	A	E	B	C		2:	(D	A)	E	(B	C)
3:	E	B	A	D	C		3:	(E	B)	(A	D	C)
4:	B	A	E	D	C		4:	B	(A | E)	D	C
5:	E	A	B	D	C		5:	(E	A	B)	(D	C)

図 11　生成する希望リストの例

通常の希望リスト（図 11 左）に従来の配属方法を，同順位リスト（図 11 右）に同順位アルゴリズムを適用し，それぞれに配属結果を得る．

次節では，この 2 種類の配属結果をどう比較するか具体例を用いて説明する．

5.2　配属結果の比較

従来の配属方法と同順位アルゴリズムとでは，そもそも希望リストの種類が異なるため，各学生の配属先研究室の希望順位がそれぞれ何位であったかという基準では比較にならない．そこで，両者の配属結果を同順位リスト上で比較し，各学生の配属先研究室の希望順位が上がったのか，もしくは下がったのか，または変わらなかったのかという順位変動の様子に注目し比較を行う．

図 12 の例は，前節で述べた生成方法による希望リストの一例で，それぞれ配属を行った結果を示している．

1:	Ⓐ	B	E	C	D	A:	1		1:	(Ⓐ	B)	E	C	D	A:	1
2:	Ⓑ	A	D	C	E	B:	2		2:	Ⓑ	(A	D	C	E)	B:	2
3:	A	Ⓓ	E	B	C	C	4		3:	A	(D	Ⓔ)	(B	C)	C	5
4:	D	Ⓒ	A	B	E	D:	3		4:	Ⓓ	(C	A)	(B	E)	D:	4
5:	Ⓔ	B	D	A	C	E:	5		5:	E	(B	D)	A	Ⓒ	E:	3

図 12　順位変動の様子の例

両者を比較すると，まず学生 3 が D → E と同順位の研究室間を移動し，それによって学生 5 が E → C と希望順位を下げ，その結果学生 4 が C → D と希望順位を上げていることが分かる．このように，従来の配属方法による配属結果と同順位アルゴリズムによる配属結果とを比較すると，一回の配属操作につき，希望順位の上がる学生と希望順位の下がる学生の両方が生じる．

希望順位の下がる学生が生じることは，一見ネガティブな印象を与えるかもしれないが，その配属結果が，あくまで同順位リストを用いた配属における安定性の定義に基づいたものであることを強調しておきたい．図 12 の例でも，希望順位が下がった学生 5 は希望順位が上がった学生 4 よりも成績下位であるため，この配属結果が安定であると確認できる．従来の配属方法による配属結果と比較して配属先研究室の希望順位が下がった学生が生じた場合には，その学生より成績上位の学生の希望順位が必ず上がっていると言うこともできる．

しかし，いくら安定な配属であるとは言え，全体として希望順位の上がる学生の人数が多いと希望順位の下がる学生の人数も多かったり，学生の成績順位層によって極端に希望順位の上がる場合と下がる場合に差があったりというようなことがあれば，同順位アルゴリズムが学生全体にとってより良い配属結果をもたらすとは言い難くなってしまう．

そこで本研究では，両者の配属結果を比較し以下の 2 種類の評価を行う．

1. 希望順位の上がる学生と下がる学生の人数の相関関係
2. 希望順位の上がる学生と下がる学生の成績の分布

「希望リストの生成→それぞれに配属を実施→配属結果の比較」という一連の流れを 10000 回繰り返し，一回の試行毎に，順位変動のあった学生の人数とその成績の分布を集計する．

図 12 の例はサイズ 5 の希望リストであるが，実際はより多人数での研究室配属を想定して，サイズ 180 のものを使用する．次節では，この 2 種類の評価の結果を，各研究室に割り当てた重みの種類ごとに示す．

5.3　シミュレーション結果

まず，研究室の人気に偏りがない状況を想定し，次の表 1 のように各研究室に重みを均等に割り当てた場合の結果を図 13 と図 14 に示す．

表 1　重みづけ 1

研究室	1	2	3	・・・	180
重み	1	1	1	・・・	1

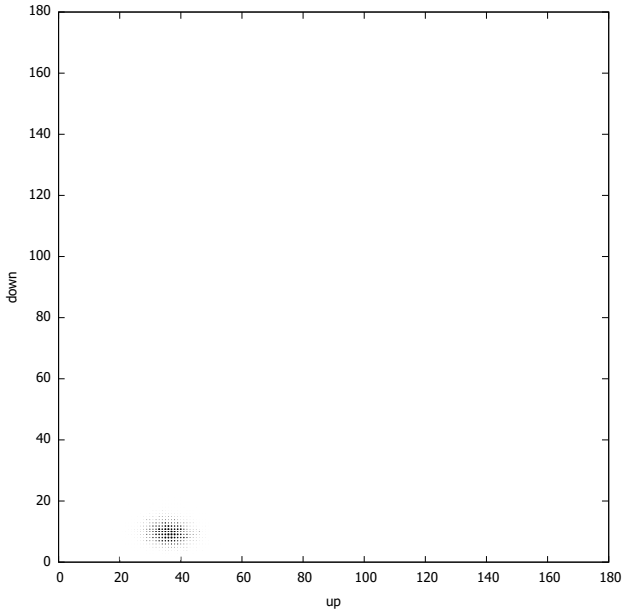

図 13　人数の相関 1

図 13 では，横軸を配属先研究室の希望順位が上がった学生の人数，縦軸を希望順位が下がった学生の人数としてその頻度を黒い円の大きさで表している．具体的には，円の半径が頻度と比例している．図から，希望順位の上がった学生が 30 人〜45 人前後に多く分布しているのに対し．希望順位の下がった学生は 5 人〜15 人前後に多く分布している．また，その分布の様子から両者に相関はないことが見て取れる．

図 14 では，横軸を学生の成績順位とし，縦軸で希望順位の上がった回数，下がった回数，変わらなかった回数の割合を示している．希望順位の上がった場合が灰色，変わらなかった場

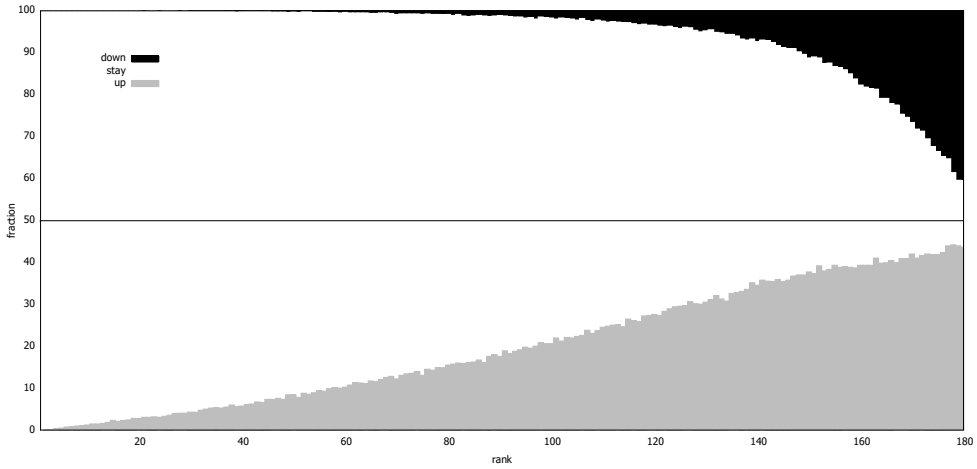

図 14　成績の分布 1

合が白，下がった場合が黒である．図から，希望順位が上がる割合は成績順位が下がるにつれて徐々に大きくなっていき，希望順位が下がる割合は，成績順位中位から下位の学生で大きくなっていることが分かる．全体としては，どの成績順位でも希望順位が下がる割合に対して希望順位が上がる割合が多く，成績順位最下位の学生でもほぼ半々の割合となっている．

次に，研究室の人気が偏っている状況を想定し，各研究室の重みに差をつけた場合の結果を示す．図 15 と図 16 は，重みを表 2 のように人気の高い研究室と人気の低い研究室を明確に分けるようにのせた場合の結果である．

表 2　重みづけ 2

研究室	1	2	3	・・・	30	31	32	・・・	180
重み	10	10	10	・・・	10	1	1	・・・	1

図 15 の人数の相関は，図 13 の人気の偏りがない場合とさほど変わらないが，図 16 の成績の分布は，成績順位中上位の学生の希望順位が上がった割合が増え，その分，希望順位の下がった学生の割合が全体として若干増えている様子が分かる．

最後に，研究室全体として人気の偏りをなだらかに表現するよう，重みを表 3 のようにのせた場合の結果を図 17 と図 18 に示す．

この場合も，図 17 の人数の相関にあまり大きな変化は見られないが，図 18 の成績の分布では，人気の偏りがない図 14 と比較して，成績順位中位から下位の学生に変化がみられる．以上の結果から，各研究室に人気の偏りがある場合でも，希望順位の上がる学生と下がる学生の

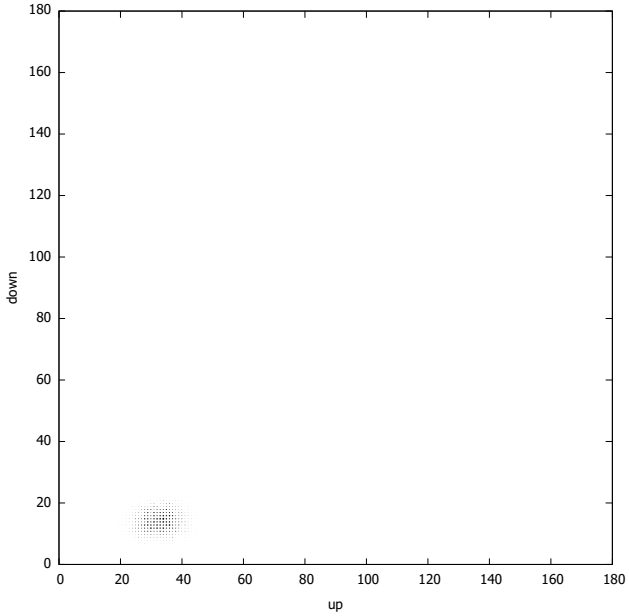

図 15　人数の相関 2

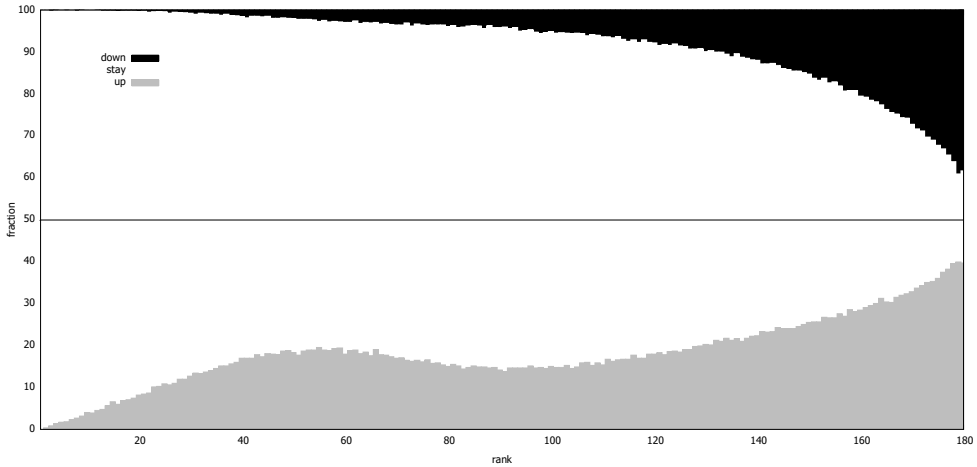

図 16　成績の分布 2

表 3　重みづけ 3

研究室	1	・・・	10	11	・・・	20	21	・・・	171	・・・	180
重み	1	・・・	1	2	・・・	2	3	・・・	18	・・・	18

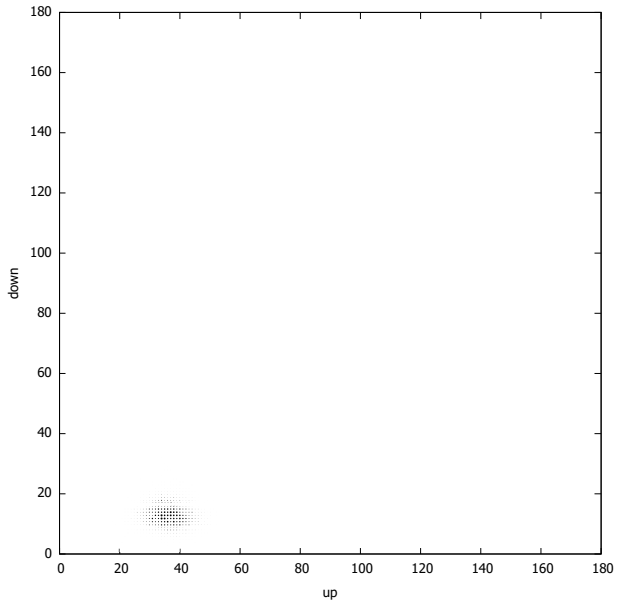

図 17　人数の相関 3

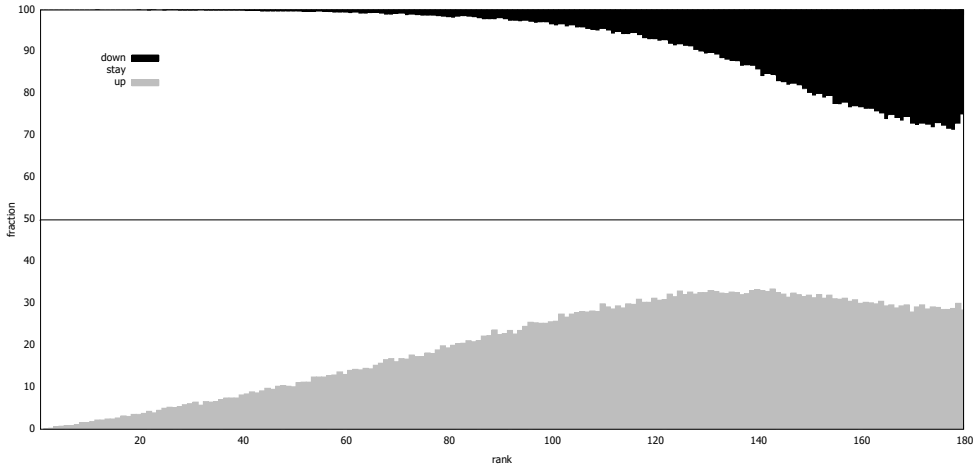

図 18　成績の分布 3

人数に相関はなく，成績の分布には多少変化が見られるが，全成績順位層で希望順位の上がる割合が多いと言える．

ここで行った研究室配属シミュレーションでは，一般に一対多のマッチングである研究室配属を一対一のマッチングと想定し，さらに同順位の “( )”を完全にランダムに配置しているため，現実的な研究室配属において，必ずしも似たような順位変動の傾向が得られるとは言い切れない．しかし，3 種類の結果いずれの場合においても，希望順位の上がる学生の割合が希望順位の下がる学生の割合に比べ多いことから，同順位リストの導入によってより高い希望が実現し，その恩恵を受ける学生のほうが全体として多そうだということが分かった．また，図 14 〜図 18 の成績の分布から，ある特定の成績順位層の学生にとって有利不利な結果になるというような傾向も見られない．以上のことから，同順位アルゴリズムは従来の配属方法と比べてより良い配属結果をもたらしそうだと言える．

5.4 計算量の評価

最後に，本アルゴリズムの計算量について簡単に述べる．まず，同順位アルゴリズムが必要とするメモリの容量は，$n \times n$ の希望リスト（n は学生の総数）が支配的な項に対応することから，$\mathcal{O}(n^2)$ であると言える．

次に，計算時間であるが，同順位アルゴリズムは手続き search を繰り返し呼び出して配属を行っていくアルゴリズムであるため，その呼び出し回数に着目することで検討を行っていく．手続き search の呼び出し回数は，玉突き操作がどの程度起こるかに依り，また玉突き操作がどの程度起こるかは，希望リスト上で同順位となっている研究室の数に依る．学生の提出する希望リストがどうなるかは誰にも分からないため，手続き search の呼び出し回数を正確に見積もることはできない．よって，最悪ケースを想定し計算時間を考える．成績順位 k のある学生（学生 k と呼ぶ）一人の配属に対し，手続き search が最も多く呼び出される場合は，学生 k の希望リストで左から $k-1$ 番目までの全ての研究室に対し手続き search を $k-1$ 回呼び出した（既に配属されている $k-1$ 人の学生全員に対し玉突き操作による移動を試みた）が，いずれの研究室にも配属できず，最終的に左から k 番目の研究室に配属される場合である．この場合，手続き earch は $(k-1)^2$ 回呼び出されると言える（アルゴリズムの構成上，正確には $k(k-1)+1$ 回）．

こうした最悪な場合が学生全員に起こることは，どのような希望リストでもあり得ないため，かなり大雑把な見積もりになってしまうが，もしそうなったとすると手続き search の総呼び出し回数は

$$\sum_{k=1}^{n}\{k(k-1)+1\} = \frac{2n^3+4n}{6} = \frac{n^3+2n}{3}$$

となり，計算時間は高々 $\mathcal{O}(n^3)$ と考えられる．

本研究で扱ってきた，学生の希望リストにのみ同順位を認めたマッチング問題が，2.3 節で述べた，男女双方の希望リストに同順位を認めた既存の研究におけるマッチング問題と比べてより簡単な問題であることを考えると，この計算時間の評価はあまり妥当とは言えない（既存の研究にける強安定マッチングアルゴリズムの最悪ケースの計算時間は $\mathcal{O}(n^2)$[4]）．

このような評価になった理由として，その評価方法がそもそも荒いものであることに加え，同順位アルゴリズム自体に無駄が含まれている可能性が考えられる．具体的には，同順位アルゴリズムは，ある学生の配属の際に行った玉突き操作で，それがうまくいかないと分かったものでも，また別の学生の配属の際に全く同じ玉突き操作を試みる場合があるため，そういった点が計算時間の評価に少なからず影響していると考えられる．

6　同順位アルゴリズムの拡張

これまで，一般に一対多のマッチングである研究室配属を一対一のマッチングとして扱ってきたが，同順位アルゴリズムを一対多のマッチングに適用することも可能である．ここでは，その具体的な方法を説明する．同順位アルゴリズムを一対多のマッチング（各研究室の定員が複数人）に適用する場合には，学生側の希望リストを拡大して使用する．この例を次の図 19 と図 20 に示す．

1:　B　(A　C)
2:　A　B　C
3:　(A　C)　B
4:　(C　B　A)
5:　(C　A)　B
6:　A　(B　C)
7:　A　(C　B)
8:　(B　C)　A

図 19　拡大前

図 19 は拡大前の学生側の希望リストである．各研究室の定員を，研究室 A が 3 人，研究室 B が 2 人，研究室 C が 3 人とする．よって計 8 人の学生を配属することになるが，学生の人数が 8 に対して研究室の数が 3 であるため，この希望リストのままでは同順位アルゴリズムを適用できない．そこで図 20 に示すように，学生の希望リスト上の各研究室をその定員の数だけ拡大して新たな希望リストとする．各研究室の空席に学生を一人ずつ割り当てていくイメージである．こうすることで，学生側の希望リストが一対一のマッチングと同様の形となり，同順位アルゴリズムがそのまま適用できるようになる．一つの研究室に対する希望に順位付けが

1: B_1 B_2 $(A_1$ A_2 A_3 C_1 C_2 $C_3)$
2: A_1 A_2 A_3 B_1 B_2 C_1 C_2 C_3
3: $(A_1$ A_2 A_3 C_1 C_2 $C_3)$ B_1 B_2
4: $(C_1$ C_2 C_3 B_1 B_2 A_1 A_2 $A_3)$
5: $(C_1$ C_2 C_3 A_1 A_2 $A_3)$ B_1 B_2
6: A_1 A_2 A_3 $(B_1$ B_2 C_1 C_2 $C_3)$
7: A_1 A_2 A_3 $(C_1$ C_2 C_3 B_1 $B_2)$
8: $(B_1$ B_2 C_1 C_2 $C_3)$ A_1 A_2 A_3

図 20　拡大後

されているように見えてしまっている部分（例えば学生 1 の B_1 と B_2）もあるが，同じ研究室を希望していることに変わりはない（B_1 と B_2 のどちらとペアになっても同じである）ため，問題はない．

7　まとめ

本研究では，安定結婚問題をベースとした従来の研究室配属方法に同順位リストを導入し，新たな研究室配属アルゴリズムを考案した．そして，このアルゴリズムが，従来の配属方法と同様に配属結果の安定性を保証しながら，学生の希望リスト作成の負担を軽減するものであることを示した．また，従来の配属方法による配属結果と同順位リストを導入した配属方法による配属結果とを比較する研究室配属シミュレーションから，同順位リストの導入は，多くの学生の配属先研究室の希望順位を上げ，全体としてより良い配属結果をもたらしそうだということが分かった．一方計算時間の評価から，同順位アルゴリズムが無駄のあるアルゴリズムとなってしまっている可能性が考えられるため，より正確な評価と併せて，これを詳しく検討していくことが今後の課題である．

謝辞

本研究に際して，様々なご指導を頂きました指導教員の西新幹彦先生に深く感謝申し上げます．

参考文献

[1] 宮崎修一,「安定結婚問題」, 京都大学 学術情報メディアセンター, 電子情報通信学会誌, Vol.88, No.3, pp.195–199 , March. 2005.

[2] 宮崎修一,「安定マッチング問題」, 京都大学 学術情報メディアセンター, 情報処理, Vol.54, No.10, pp.1064–1071, Oct. 2013.

[3] D. Gale and L. S. Shapley, “College Admissions and the Stability of Marriage”, The American Mathemarical Monthly , Vol. 69, No. 1 (Jan., 1962), pp.9–15.

[4] Robert W. Irving, “Stable marriage and indifference”, Discrete Applied Mathematics, no.48, pp.261–272, 1994.

付録 A　ソースコード

A.1　従来の配属方法を実装したプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int labTotal;
int studentTotal;

struct prefList_normal{
    int *state;
    int *pref;
};

struct prefList_normal *studentList_normal;
int *engageList_normal;

int search_normal(int student, int option); //recursive function
int createList_normal(FILE *fp);

int *state;
int *pref;
struct prefList_normal *student;

int normal(){
    int size;

    FILE *fp;
    char fname[] = "abc4.txt";
    fp = fopen(fname, "r");

    if(fscanf(fp, "%d", &size) == EOF){
        printf("file error");
    }
    labTotal = size;
    studentTotal = size;

    studentList_normal = (struct prefList_normal *)malloc(sizeof(struct prefList_normal) * studentTotal);
    engageList_normal = (int *)malloc(sizeof(int) * labTotal);
    for (int n = 0; n < labTotal; n++) { //-1 埋め
        engageList_normal[n] = -1;
    }

    if(createList_normal(fp) == -1){ //create preference-list
        printf("file error\n");
    }

    //start
    int key;
    int i;
    for(i = 0; i < studentTotal; i++){
        key = 0;
        while(search_normal(i, studentList_normal[i].pref[key]) == 0){
            studentList_normal[i].state[key] = -1;
            key++;
        }
        engageList_normal[studentList_normal[i].pref[key]] = i;
    }

    //output
    FILE *fpr;
    fpr = fopen("result.txt", "w");
    for(int j = 0; j < labTotal; j++){
```

```
        fprintf(fpr, "%d ", engageList_normal[j]);
    }
    fclose(fpr);
    fclose(fp);

    for(int n = 0; n < studentTotal; n++){
        free(studentList_normal[n].state);
        free(studentList_normal[n].pref);
    }

    free(engageList_normal);
    free(studentList_normal);
    return(0);
}

int search_normal(int student, int option){

    if(engageList_normal[option] == -1){
        return(1);
    }
    else{
        return(0);
    }
}

int createList_normal(FILE *fp){
    int studentID = 0;
    int key = 0;
    int input;
    int flag = 0;

    if(fp == NULL) {
        return(-1);
    }

    while(studentID < studentTotal){

        state = (int *)malloc(sizeof(int) * labTotal);
        for (int i = 0; i < labTotal; i++) { //0 埋め
            state[i] = 0;
        }

        pref = (int *)malloc(sizeof(int) * labTotal);

        while(key < labTotal){
            if(fscanf(fp, "%d", &input) == EOF){
                return(-1);
            }
            if(input == -1){
                if(flag == 0){
                    flag = 1;
                }
                else{
                    flag = 0;
                }
            }
            else{
                pref[key] = input;
                key++;
            }
            if((key == labTotal)&&(flag == 1)){ //最後の余分な-1 を読み込み
                if(fscanf(fp, "%d", &input) == EOF){
                return(-1);
                }
            }
        }
        studentList_normal[studentID].state = state;
        studentList_normal[studentID].pref = pref;
```

```
        studentID++;
        key = 0;
        flag = 0;
    }

    fclose(fp);

    return(0);
}
```

A.2　同順位アルゴリズムを実装したプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern int labTotal;
extern int studentTotal;

struct prefList_tie{
    int *state;
    int *order;
    int *pref;
    int lock;
};

struct prefList_tie *studentList;
int *engageList_tie;

int search_tie(int student, int option); //recursive function
int createList_tie(FILE *fp);

int *state;
int *order;
int *pref;
struct prefList_tie *student;

int tie(){
    int size;

    FILE *fp;
    char fname[] = "abc4.txt";
    fp = fopen(fname, "r");

    if(fscanf(fp, "%d", &size) == EOF){
        printf("file error");
    }
    labTotal = size;
    studentTotal = size;

    studentList = (struct prefList_tie *)malloc(sizeof(struct prefList_tie) * studentTotal);
    engageList_tie = (int *)malloc(sizeof(int) * labTotal);
    for (int n = 0; n < labTotal; n++) { //-1 埋め
        engageList_tie[n] = -1;
    }

    if(createList_tie(fp) == -1){ //create preference-list
        printf("file error\n");
    }

    //start
    int key;
    int i;
    for(i = 0; i < studentTotal; i++){
        key = 0;
```

```
        while(search_tie(i, studentList[i].pref[key]) == 0){
            studentList[i].state[key] = -1;
            key++;
        }
        engageList_tie[studentList[i].pref[key]] = i;
    }

    //output
    FILE *fpr;
    fpr = fopen("result.txt", "a");
    fprintf(fpr, "\n");
    for(int j = 0; j < labTotal; j++){
        fprintf(fpr, "%d ", engageList_tie[j]);
    }
    fclose(fpr);
    fclose(fp);

    for(int n = 0; n < studentTotal; n++){
        free(studentList[n].state);
        free(studentList[n].order);
        free(studentList[n].pref);
    }

    free(studentList);
    free(engageList_tie);

    return(0);
}

int search_tie(int student, int option){
    if(studentList[student].lock == 1){
        return(0);
    }
    studentList[student].lock = 1;

    int key = 0;
    int target;
    int order;

    if(engageList_tie[option] == -1){
        studentList[student].lock = 0;
        return(1);
    }

    else{
        while(studentList[engageList_tie[option]].pref[key] != option){
            key++;
        }
        target = key;
        if(key != 0){
            while((studentList[engageList_tie[option]].order[key - 1] ==
                  studentList[engageList_tie[option]].order[key])&&(key > 0)){
                key--;
            }
        }
        order = studentList[engageList_tie[option]].order[key];
        while(studentList[engageList_tie[option]].order[key] == order){
            if((studentList[engageList_tie[option]].state[key] == 0)&&
               (studentList[engageList_tie[option]].pref[key] != option)){
                   if(search_tie(engageList_tie[option],
                      studentList[engageList_tie[option]].pref[key]) == 1){
                        engageList_tie[studentList[engageList_tie[option]].pref[key]] = engageList_tie[option];
                        studentList[student].lock = 0;
                        return(1);
                    }
            }
            key++;
        }
        studentList[student].lock = 0;
```

```
        return(0);
    }
}

int createList_tie(FILE *fp){
    int studentID = 0;
    int key = 0;
    int input;
    int tempOrder = 1;
    int flag = 0;

    if(fp == NULL) {
        return(-1);
    }

    while(studentID < studentTotal){

        state = (int *)malloc(sizeof(int) * labTotal);
        for (int i = 0; i < labTotal; i++) { //zero 埋め
            state[i] = 0;
        }

        order = (int *)malloc(sizeof(int) * labTotal);

        pref = (int *)malloc(sizeof(int) * labTotal);

        while(key < labTotal){
            if(fscanf(fp, "%d", &input) == EOF){
                return(-1);
            }
            if(input == -1){
                if(flag == 0){
                    flag = 1;
                }
                else{
                    flag = 0;
                    tempOrder = key + 1;
                }
            }
            else{
                if(flag == 1){
                    order[key] = tempOrder;
                    pref[key] = input;
                }
                else if(flag == 0){
                    order[key] = tempOrder;
                    pref[key] = input;
                    tempOrder++;
                }
                key++;
            }
            if((key == labTotal)&&(flag == 1)){ //最後の余分な-1 を読み込み
                if(fscanf(fp, "%d", &input) == EOF){
                return(-1);
                }
            }
        }
        studentList[studentID].state = state;
        studentList[studentID].order = order;
        studentList[studentID].pref = pref;
        studentList[studentID].lock = 0;

        studentID++;
        tempOrder = 1;
        key = 0;
        flag = 0;
    }

    fclose(fp);
```

```
    return(0);
}
```

A.3 重みを設定し研究室配属シミュレーションを行うプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define roop 10000
#define size 180

int date[size + 1][size + 1];
int up_perf[size];
int down_perf[size];

int sample(int desigSize, int *desigWeight);
int normal();
int tie();
int analyze();

extern int up;
extern int down;

int main(void) {
    srand((unsigned int)time(NULL));

    /*
    int weight[] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
    */

    /*
    int weight[] = {10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
                    10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
    */


    int weight[] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                    3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
                    5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
                    7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
                    9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
                    11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
                    13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
                    15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
                    17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18};
```

```
    FILE *fpr;
    FILE *fprr;

    int tempUp;
    int tempDown;

    for (int n = 0; n < (size + 1); n++){ //0 埋め
        for (int l = 0; l < (size + 1); l++){
            date[n][l] = 0;
        }
    }

    for(int i = 0; i < roop; i++){

        sample(size, weight);
        normal();
        tie();
        analyze();

        date[up][down]++;

        printf("%d,%d,%d\n", i, up, down);
    }

    fpr = fopen("distdate.txt", "w");
    for(int m = 0; m < (size + 1); m++){
        for(int k = 0; k < (size + 1); k++){
            fprintf(fpr, "%d %d %d\n", m, k, date[m][k]);
        }
    }
    fclose(fpr);

    int sum = 0;
    int sum2 = 0;
    fprr = fopen("perf_distdate.txt", "w");
    for(int p = 0; p < size; p++){
        fprintf(fprr, "%d %d %d %d\n", p + 1, up_perf[p],
                (roop - (up_perf[p] + down_perf[p])), down_perf[p]);
        sum = sum + up_perf[p];
        sum2 = sum2 + down_perf[p];
    }
    fclose(fprr);
    printf("%d,%d", sum, sum2);

    return (0);
}
```

A.4　希望リストをランダムに生成するプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct status{
    int labNum;
    int weight;
    int flag;
};

void shuffle(int* array, int size, struct status* weightList, int weightTotal);

int sample(int desigSize, int *desigWeight){
    int flag = 0;
    int par = -1;
```

```
    FILE *fp;
    fp = fopen("abc4.txt", "w");

    int size;
    size = desigSize;

    fprintf(fp, "%d", size);
    fprintf(fp, "\n");

    struct status *weightList;
    weightList = (struct status *)malloc(sizeof(struct status) * size);

    int weight;
    int weightTotal = 0;
    for(int n = 0; n < size; n++){
        weight = desigWeight[n];
        weightList[n].labNum = n;
        weightList[n].weight = weight;
        weightList[n].flag = 0;
        weightTotal = weightTotal + weight;
    }

    for(int i = 0; i < size; i++){

        int array[size];

        shuffle(array, size, weightList, weightTotal);

        for(int k = 0; k < size; k++){
            if(flag == 0){
                if((rand() % 2 == 1)&&(k <= size -2)){
                    fprintf(fp, "%d ", par);
                    flag = 1;
                }
            }
            else if(flag == 1){
                flag = -1;
            }
            else if(flag == -1){
                if(rand() % 2 == 1){
                    fprintf(fp, "%d ", par);
                    if((rand() % 2 == 1)&&(k <= size -2)){
                        fprintf(fp, "%d ", par);
                        flag = 1;
                    }
                    else{
                        flag = 0;
                    }
                }
            }
            fprintf(fp, "%d ", array[k]);
        }
        if(flag == -1){
            fprintf(fp, "%d ", par);
        }
        flag = 0;
        fprintf(fp, "\n");
    }

    fclose(fp);
    free(weightList);

    return(0);
}

void shuffle(int* array, int size, struct status* weightList, int weightTotal){
    int tempWeightTotal = weightTotal;
    for(int i = 0; i < size; i++){
        int j = 0;
```

```
        int target = (rand() % tempWeightTotal) + 1;
        int sum = 0;
        while(j < size){
            if(weightList[j].flag == 0){
                sum = sum + weightList[j].weight;
                if(target <= sum){
                    array[i] = weightList[j].labNum;
                    weightList[j].flag = 1;
                    tempWeightTotal = tempWeightTotal - weightList[j].weight;
                    break;
                }
            }
            j++;
        }
    }
    for(int k = 0; k < size; k++){
        weightList[k].flag = 0;
    }
}
```

A.5　配属結果を分析するプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int labTotal;
int studentTotal;

struct prefList{
    int *state;
    int *order;
    int *pref;
    int lock;
};

struct prefList *studentList;
int *normal;
int *tie;

int createList(FILE *fp);

int up;
int down;

extern int up_perf[];
extern int down_perf[];

int *state;
int *order;
int *pref;

int analyze(){
    int size;

    FILE *fp;
    char fname[] = "abc4.txt";
    fp = fopen(fname, "r");

    if(fscanf(fp, "%d", &size) == EOF){
        printf("file error");
    }
    labTotal = size;
    studentTotal = size;
```

```
studentList = (struct prefList *)malloc(sizeof(struct prefList) * studentTotal);

if(createList(fp) == -1){ //create preference-list
    printf("file error\n");
}

//create rezult-list
normal = (int *)malloc(sizeof(int) * studentTotal);
tie = (int *)malloc(sizeof(int) * studentTotal);

FILE *fpr;
fpr = fopen("result.txt", "r");

int key = 0;
int input;
while(key < labTotal * 2){
    if(fscanf(fpr, "%d", &input) == EOF){
        return(-1);
    }
    else{
        if(key < labTotal){
            normal[key] = input;
        }
        else{
            tie[key - labTotal] = input;
        }
    }
    key++;
}

//analyze
int targetStu;
int ind;
int sub;
int compVal;
up = 0;
down = 0;
for(int i = 0; i < labTotal; i++){
    sub = 0;
    targetStu = normal[i];
    while(studentList[targetStu].pref[sub] != i){
        sub++;
    }
    compVal = studentList[targetStu].order[sub];

    ind = 0;
    sub = 0;
    while(tie[ind] != targetStu){
        ind++;
    }
    while(studentList[targetStu].pref[sub] != ind){
        sub++;
    }

    if(compVal > studentList[targetStu].order[sub]){
        up++;
        up_perf[targetStu]++;
    }
    else if(compVal < studentList[targetStu].order[sub]){
        down++;
        down_perf[targetStu]++;
    }
}

fclose(fp);
fclose(fpr);

for(int n = 0; n < studentTotal; n++){
    free(studentList[n].state);
```

```
        free(studentList[n].order);
        free(studentList[n].pref);
    }

    free(studentList);
    free(normal);
    free(tie);

    return(0);
}

int createList(FILE *fp){
    int studentID = 0;
    int key = 0;
    int input;
    int tempOrder = 1;
    int flag = 0;

    if(fp == NULL) {
        return(-1);
    }

    while(studentID < studentTotal){

        state = (int *)malloc(sizeof(int) * labTotal);
        for (int i = 0; i < labTotal; i++) { //zero 埋め
            state[i] = 0;
        }

        order = (int *)malloc(sizeof(int) * labTotal);

        pref = (int *)malloc(sizeof(int) * labTotal);

        while(key < labTotal){
            if(fscanf(fp, "%d", &input) == EOF){
                return(-1);
            }
            if(input == -1){
                if(flag == 0){
                    flag = 1;
                }
                else{
                    flag = 0;
                    tempOrder = key + 1;
                }
            }
            else{
                if(flag == 1){
                    order[key] = tempOrder;
                    pref[key] = input;
                }
                else if(flag == 0){
                    order[key] = tempOrder;
                    pref[key] = input;
                    tempOrder++;
                }
                key++;
            }
            if((key == labTotal)&&(flag == 1)){ //最後の余分な-1 を読み込み
                if(fscanf(fp, "%d", &input) == EOF){
                return(-1);
                }
            }
        }
        studentList[studentID].state = state;
        studentList[studentID].order = order;
        studentList[studentID].pref = pref;
        studentList[studentID].lock = 0;
```

```
        studentID++;
        tempOrder = 1;
        key = 0;
        flag = 0;
    }

    fclose(fp);

    return(0);
}
```