

Title	エージェントベースモデルによる飲食業界における労働市場のシミュレーション分析
Sub Title	
Author	徐, 晃生(Seo, Hwangsaeng) 高橋, 大志(Takahashi, Hiroshi)
Publisher	慶應義塾大学大学院経営管理研究科
Publication year	2019
Jtitle	
JaLC DOI	
Abstract	
Notes	修士学位論文. 2019年度経営学 第3575号
Genre	Thesis or Dissertation
URL	https://koara.lib.keio.ac.jp/xoonips/modules/xoonips/detail.php?koara_id=KO40003001-00002019-3575

慶應義塾大学学術情報リポジトリ(KOARA)に掲載されているコンテンツの著作権は、それぞれの著作者、学会または出版社/発行者に帰属し、その権利は著作権法によって保護されています。引用にあたっては、著作権法を遵守してご利用ください。

The copyrights of content available on the Keio Associated Repository of Academic resources (KOARA) belong to the respective authors, academic societies, or publishers/issuers, and these rights are protected by the Japanese Copyright Act. When quoting the content, please follow the Japanese copyright act.

慶應義塾大学大学院経営管理研究科修士課程

学位論文（ 2019 年度）

論文題名

エージェントベースモデルによる飲食業界における労働市場のシミュレーション分析

主 査	高橋 大志
副 査	大林 厚臣
副 査	大藪 毅

氏 名	徐 晃生
-----	------

論文要旨

所属ゼミ	高橋研究室	氏名	徐 晃生
(論文題名) エージェントベースモデルによる飲食業界における労働市場のシミュレーション分析			
(内容の要旨) はじめに 企業経営において労働力の確保は主要な関心事の一つに挙げられる。日本において、人口動態の変化などを背景として人手不足は深刻化しており、とりわけ飲食業界においては恒常化しつつある課題である。本研究では、人手不足が最も深刻な飲食業界の企業に焦点をあて、飲食企業と非正規労働者の労働市場におけるマッチングプロセスの観点からエージェントベースモデルを用いた分析を行う。これらの分析を通じ、飲食業界の労働市場全体の特徴を整理し、現状の問題点とその改善点を示すことを目的とする。 分析方法 本分析において、飲食業界の労働市場内部のジョブマッチングプロセスを観察するために、エージェントベースモデルを構築する。S4 Simulation System (エスクワトロシミュレーションシステム) を利用し、コンピュータ上に現状の飲食業界の労働市場を模した仮想空間を作り、非正規雇用者と飲食企業の求人・求職・応募・採用・離職などの行動をシミュレーションにより表現する。 労働市場モデル内には労働者エージェントと企業エージェントの2種類が存在し、労働者エージェントと企業エージェントはそれぞれレベルを持っている。労働者のレベルは企業にとっての魅力度(企業から見た能力や個性など)であり、企業のレベルは労働者にとっての魅力度(賃金)である。したがって、労働者エージェントのレベルは飲食店舗責任者によるアンケート結果を用いて設定し、企業エージェントのレベルは賃金別飲食店舗数の分布を用いて設定した。 シナリオ分析 シナリオごとに労働市場モデルを構築し、シミュレーションを行う。比較対象となるシナリオは4つである。シナリオ1は飲食業界の労働市場の現状を表したもので、求人倍率を3.6倍にした。シナリオ2は外部からの影響がないまま人手不足が解消した場合を表現しており、全労働者エージェントを増やし、有効求人倍率を1.1倍に設定した。シナリオ3は外国人労働者の増加により人手不足が緩和された状態を表しており、外国人労働者エージェントを増やし、有効求人倍率を2.4倍に設定した。シナリオ4は外国人労働者がさらに増加した状態であり、全体に占める外国人労働者の割合が相対的に大きい(72%)と、外国人労働者の魅力度は全体の相対的魅力度に類似するため、外国人労働者のレベルの幅を(1~4)に修正し、有効求人倍率を1.1倍に設定した。 分析結果 求人充足率と就労率の二つの視点で、労働市場全体の最適化を考えた場合、外国人労働者の全体に占める割合を増やし、人手不足を解消すべきであるという結論に達した。現状シナリオの分析から現在の飲食企業の労働市場の特徴として、労働者の高い就労率が挙げられるが、その一方で企業の低すぎる求人充足率も明らかとなった。外国人労働者の割合増加による人手不足解消は、現在の高就労率を保ったまま、企業の求人充足率も上げることが可能である。しかし、外国人を増加させ続ければ労働者の賃金はいつまでも上がらない。政府の施策は企業の利益を確保または延命させるほかない。 まとめ 飲食企業の労働市場は、非正規雇用者によって支えられているという点で他産業とは異なる特徴を有しており、現在最も問題を抱えている労働市場の一つである。本研究では、この課題を解決するために、飲食企業の労働市場のモデル化を行い、シナリオ分析を通じ、その特徴を示すことができた。今後の課題としては、さらにサンプルデータ量を増やしたモデルの構築、地理的要素を含むモデル化、強化学習などがあげられる。			

目次

第1章	序論.....	2
第1節	背景と目的	2
第2節	労働市場の現状	2
第3節	研究方法	5
第2章	先行研究	6
第1節	労働市場に関する研究	6
第2節	エージェントベースモデルに関する研究	6
第3章	モデル	7
第1節	労働者エージェント	7
第2節	企業エージェント	9
第3節	モデルの妥当性	11
第4章	分析結果	13
第1節	シナリオ分析	13
第2節	利益構造分析	18
第5章	考察	25
第6章	まとめ	25
	参考文献	25
	APPENDIX	26

第1章 序論

第1節 背景と目的

昨今、日本で人手不足が叫ばれるなか、企業経営において労働力の確保は主要な関心事の一つに挙げられる。実際に人手不足が原因で倒産する企業は増加傾向にあり、東京商工リサーチによると2018年度の人手不足関連倒産は前年度比28.6%増の400件で過去最多を記録した。最新の人手不足関連倒産状況（2019年上半期）の内訳では、代表者や幹部役員の死亡、病气入院、引退などによる「後継者難」型が109件（前年同期146件）、人手確保が困難で事業継続に支障が生じた「求人難」型が47件（同19件）、中核社員の独立、転職などで事業継続に支障が生じた「従業員退職」型が20件（同10件）、賃金等の人件費アップから収益が悪化した「人件費高騰」型が15件（同10件）であり、「求人難」型が2.4倍増と増加率が最大だった。このように、国内の人口動態の変化などを背景とした人手不足が企業に与える影響は深刻化しており、とりわけ飲食企業にとっては恒常化しつつある課題の一つとなっている。筆者の一人は飲食企業の経営に従事し、この危機を直に感じており、一刻も早い解決を求め本研究に取り組むこととした。

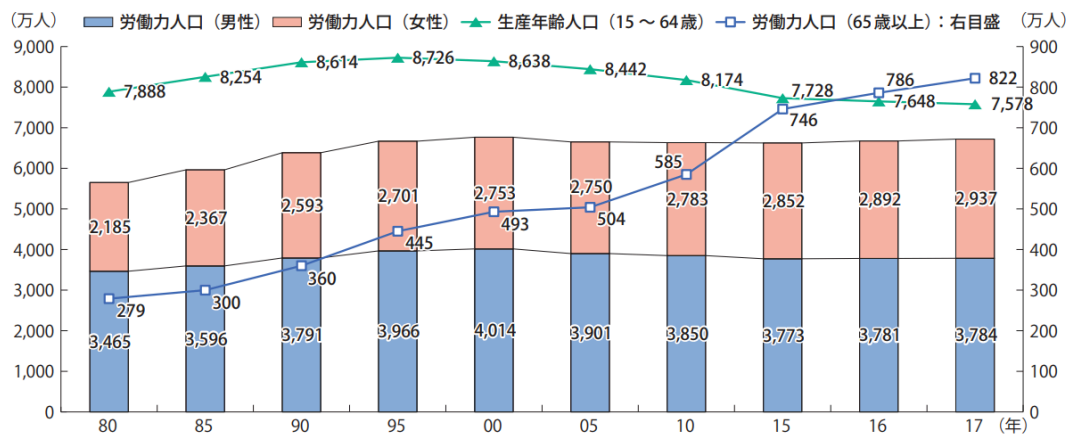
本研究では、飲食企業に焦点をあて、飲食企業と非正規労働者の労働市場におけるマッチングプロセスの観点から分析を行う。これらの分析を通じ、飲食企業の労働市場全体の特徴を整理し、現状の問題点とその改善点を示すことを目的とする。飲食店に焦点を当てた分析はいくつか報告されているが[1]、本分析ではエージェントベースモデルを用いた分析を試みる。

第2節 労働市場の現状

数多くの業種が存在する中で飲食企業が深刻な人手不足に陥る根本的な原因として、日本の人口動態（少子高齢化、労働者の構造変化）と利益構造が挙げられる。

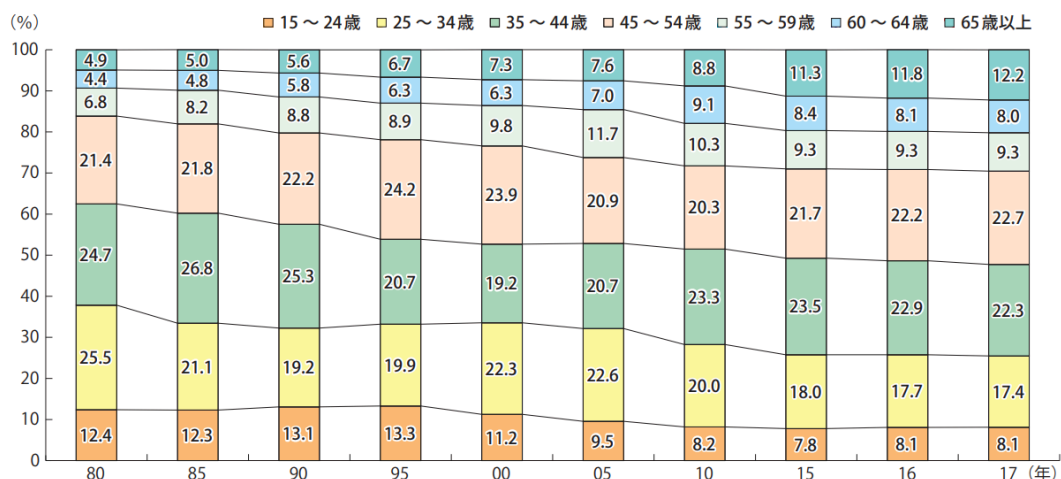
労働市場を概観する際に、生産年齢人口および年齢構成は主要な指標の一つに挙げられる。総務省が発表している「労働力調査（詳細集計・長期時系列データ）」によると、日本独自の生産年齢人口の変化と労働人口の年齢構成が人手不足に大きく影響していること確認できる。生産年齢人口が2000年の約8,600万人から2017年までに約1,000万人以上も減っており、さらに労働人口の年齢構成では、飲食企業にとって最も必要な44歳以下の労働人口（若者や中堅社員）は減少傾向となっている。一方で、55歳以上の労働人口（中でも60歳以上）は増加傾向にあるが、飲食企業での業務は肉体労働が多く、高齢者は体力的に厳しい状況にある。図1.1と図1.2は総務省「労働力調査」から引用したデー

タである。



出所 総務省 労働力調査

図 1.1 労働力人口と生産年齢人口の推移

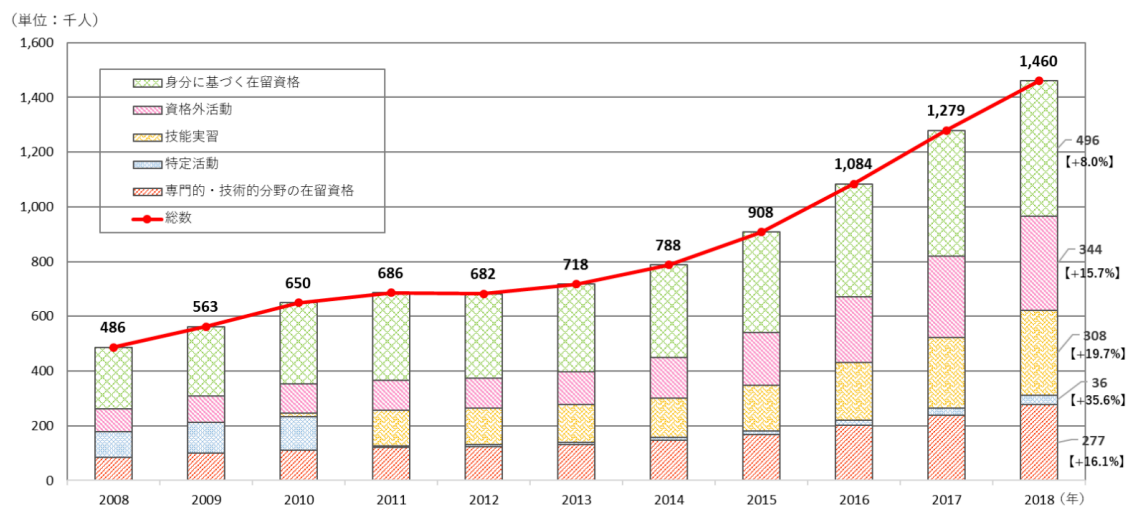


出所 総務省 労働力調査

図 1.2 労働力人口の年齢構成比の推移

政府は労働者不足の対応策の一つとして、外国人労働者の受け入れを推進しており、外された。介護、飲食、建設、飲食料品製造等の人材不足が深刻な 14 業種で就労を認める「特定技能」と呼ばれる在留資格を導入し、5 年間で最大約 35 万人の受け入れが見込まれている。

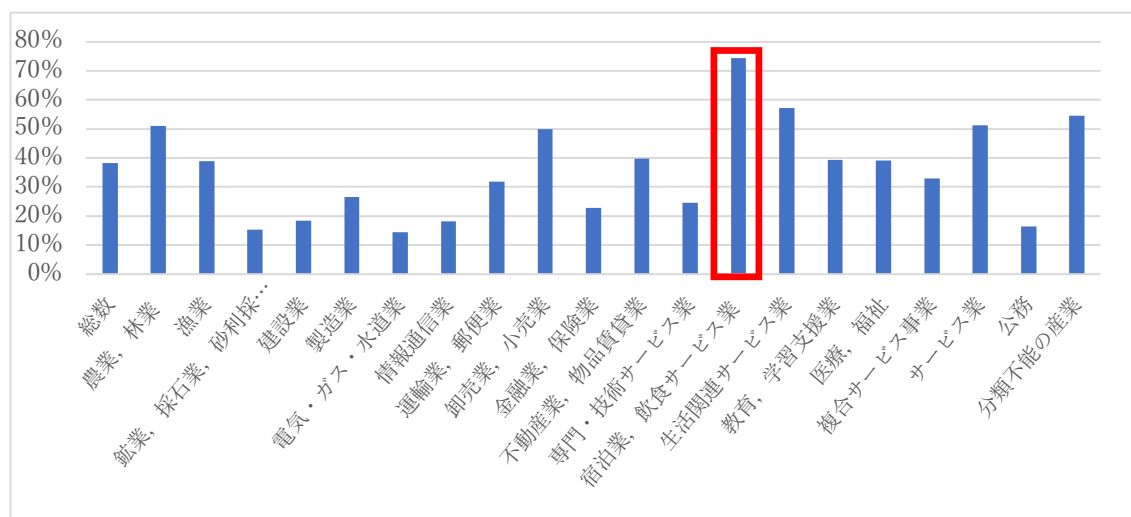
日本で働く外国人労働者の数は 5 年で倍増している。日本の労働力人口に占める割合も増えており、2018 年には全産業で 2% を超え、飲食産業では 1 割近くを外国人が占めている [4]。図 1.3 は厚生労働省「外国人雇用状況」から引用したデータであり、外国人労働者が年々増加していることがわかる。



出所 総務省 労働力調査

図 1.3 日本国内の外国人労働者の推移

非正規雇用に関する統計指標も報告されている。例えば、総務省が発表している平成 29 年就業構造基本調査によると、飲食産業の非正規雇用者比率は 75%と最も高く、飲食企業が非正規雇用者に強く依存していることを確認できる。



出所 総務省 労働力調査

図 1.4 産業別非正規雇用者比率

さらに企業の反応として、帝国バンクは「人手不足に対する企業の動向調査」で、非正規雇用者が不足していると回答した企業は、全体が 31.8%であるのに対し、飲食企業は 78.6%と最も高いとの報告を行っている（表 1-1）。飲食企業の主要な従業者である非正規雇用者を確保するためには、賃金を上げる（高い賃金で募集をかける）ことが最も有効

な手段の一つに挙げられるが、利益率が低い飲食企業ではすでに利益構造に余裕がない状況にある。東洋経済「会社四季報（2018年）」によると、飲食業界の労働者の平均年収は40歳時点で年収481万円と、63ある全業界の中で下から5番目である。このように給与水準が低いことは、人手不足の大きな要因のひとつである。給与水準が他の業界と比べて低い理由は、飲食店はあらかじめ客数を想定することが難しいことが挙げられる。さらに、複数のメニューをいつでも提供できるようにしておくため、客数に関わらず、ホールやキッチンにいる従業員は常に一定数確保する必要がある。このように人件費が多くかかりやすいにも関わらず、食材費やテナント料などかかる費用が多く、利益率が低い傾向にあるので、給与水準が低めの企業が多いといえる。

その結果、飲食産業の有効求人倍率は約3.6倍となっており、全業種の有効求人倍率が約1.5倍であることを考えると、他業種と比較して人手不足がより深刻な課題となっている[7]。

表 1-1 従業員が「不足」している企業割合

非正社員	(%)			
	2019年4月	2018年4月	2017年4月	
飲食店	↑ 78.6	↓ 77.3	－	80.0
飲食料品小売	↓ 63.9	↑ 73.1	－	60.6
人材派遣・紹介	↑ 56.3	↓ 46.5	－	60.4
メンテナンス・警備・検査	↓ 56.2	↑ 56.7	－	54.4
各種商品小売	↑ 56.1	↓ 53.5	－	58.1
娯楽サービス	↑ 54.7	↓ 50.9	－	64.8
旅館・ホテル	↑ 54.2	↓ 37.0	－	40.7
電気・ガス・水道・熱供給	↑ 50.0	→ 25.0	－	25.0
運輸・倉庫	↑ 47.5	↑ 43.9	－	41.2
飲食料品・飼料製造	↑ 46.3	↑ 44.1	－	41.7

出所 帝国バンク 人手不足に対する企業の動向調査

第3節 研究方法

本研究では、飲食業界の労働市場内部のジョブマッチングプロセスを観察するために、エージェントベースモデルを構築する。S4 Simulation System（エスクワトロシミュレーションシステム）を利用し、コンピュータ上に現状の飲食業界の労働市場を模した仮想空間を作り、非正規雇用者と飲食企業の求人・求職・応募・採用・離職などの行動をシミュレーションにより表現する。

労働市場モデル内には労働者エージェントと企業エージェントの2種類が存在し、労働者エージェントと企業エージェントはそれぞれレベルを持っている。労働者のレベルは企

業にとっての魅力度（企業から見た能力や個性など）であり、企業のレベルは労働者にとっての魅力度（賃金）である。したがって、労働者エージェントのレベルは飲食店舗責任者によるアンケート結果を用いて設定し、企業エージェントのレベルは賃金別飲食店舗数の分布を用いて設定した。

第2章 先行研究

第1節 労働市場に関する研究

非正規雇用の労働市場に関する研究は、これまで多くの研究者によって行われており、特に労働市場における労働者の能力に関する情報の非対称性の存在により、非正規雇用での就労が安定定期的な職への移行可能性を高めるという、ステッピングストーン効果の検証が進められており、仮説を支持する実証結果が現れている。

Reilly と Bothfeld (2005)によると、パートタイム労働者は配偶者の所得や税制、家事・育児といった家計内生産を前提とした労働供給を行っており、フルタイム労働を好まない傾向にあることが明らかにした。

有効求人倍率に関する研究として、増井淳(2017)は労働経済学の理論的フレームワークから人手不足のメカニズムを考察した。理論的には、外部からの影響により一時的に人手不足が発生しても、賃金の調整メカニズムにより人手不足が解消するが、解消にはある程度時間がかかることを示している。

第2節 エージェントベースモデルに関する研究

エージェントベースモデルは、1990年代に登場した、コンピュータシミュレーションを用いて、エージェント同士の相互作用から現われる、複雑な社会現象を分析・予測する手法である。エージェントとは一定のルールに従い自律的に行動する仮想的な実行主体であり、例えば、個人や団体などをエージェントとして表現することが可能である。エージェントベースモデルには、統計的アプローチや数理モデルなどの従来の方法とは異なる特徴がある。その特徴を活用した事例として、エージェントベースモデルを労働市場の分析に利用した研究を紹介する。

Neugart(2004)は、情報の非対称性が高く、抽象度が高い労働市場モデルを構築し、求人枠と求職者を結びつけるジョブマッチングの動きについてエージェントベースモデルを使って説明している。Neugart はこのようなエージェントの相互作用(interaction)により市場に生み出される動きを内生的アプローチと表現しており、従来からある統計的アプローチや数理モデルを使った分析を外部性アプローチと表現した。エージェントベースモデルを使った内生的アプローチでは、係数を前もって外部から与える必要がなく、シミュレー

ション行うことにより、外生的アプローチのパラメータと同じ値を得ることができると主張している。

第3章 モデル

本研究では、飲食企業の労働市場が直面する課題にエージェントベースモデルを通じ取り組む。図 3.1 は、コンピュータ内に構築した労働市場の実行画面の一部を示したものである。労働市場には実行主体として労働者エージェントと企業エージェントの2種類が存在する。モデル構築の際に、日本の飲食業界という特定の労働市場を表現するため、エージェントの内部状態とモデルの環境は収集した外部データによって構築するものとする。なお、本研究では、NTT データ数理システム S-quattro Simulation System を用い分析を行った。



図 3.1 分析実行画面

第1節 労働者エージェント

飲食企業へ非正規雇用の形態で求職・就労する人を労働者エージェントと置く。フリーター、学生、主婦、外国人の4種類の労働者エージェントを設定する。エージェント数は3,600 (360日×10人)とし、労働者エージェントの1/360が1日ごとに労働市場に投入され、求職活動を開始する。表1は、各労働者エージェントの割合とレベル、応募、離職率を示したものである。

各労働者エージェントの割合は、総務省「就業構造基本調査」と厚生労働省「外国人雇用状況」のデータを元に作成し、各労働者エージェントの離職率は、ディップ総合研究所「アルバイト・パートスタッフの離職事情」、株式会社ジェネックスソリューションズ「外国人離職率」のデータを元に作成した。

労働者は自身の能力を表す値として1～6のレベルを有する。各労働者（フリーター、学生、主婦、外国人）によってレベルの幅が異なるが、各レベルの人数は正規分布によって割り当てる。各労働者のレベルの幅は飲食店舗責任者30人を対象に実施したアンケートを元に設定した。

表 3-1 労働者エージェントの種類

種類	フリーター	学生	主婦	外国人
割合	3割 (30%)	4割 (37%)	2割 (24%)	1割 (9%)
レベル	2~6	3~5	2~5	1~3
応募	1/360	1/360	1/360	1/360
離職率	1ヶ月：5%	1ヶ月：13%	1ヶ月：3%	1ヶ月：10%
	2ヶ月：4%	2ヶ月：6%	2ヶ月：2%	2ヶ月：5%
	3ヶ月：4%	3ヶ月：5%	3ヶ月：2%	3ヶ月：5%
	4ヶ月：3%	4ヶ月：5%	4ヶ月：2%	4ヶ月：4%
	5ヶ月：2%	5ヶ月：5%	5ヶ月：2%	5ヶ月：3%
	6ヶ月：2%	6ヶ月：5%	6ヶ月：2%	6ヶ月：3%
	7ヶ月：3%	7ヶ月：5%	7ヶ月：2%	7ヶ月：2%
	8ヶ月：3%	8ヶ月：4%	8ヶ月：2%	8ヶ月：2%
	9ヶ月：3%	9ヶ月：4%	9ヶ月：2%	9ヶ月：1%
	10ヶ月：3%	10ヶ月：4%	10ヶ月：2%	10ヶ月：1%
	11ヶ月：3%	11ヶ月：4%	11ヶ月：2%	11ヶ月：1%
	12ヶ月：3%	12ヶ月：4%	12ヶ月：2%	12ヶ月：1%

労働者エージェントのレベルは企業にとっての魅力度であり、客観的な数値であるため労働者自身は正確に把握できない。したがって労働者は、自身のレベルよりも一つレベルを落としたレベルと同等か、あるいはそれ以上のレベルを持つ企業に魅力を感じて応募する。応募する企業は、条件にあった企業の中から1つを乱数選択により選び出される。

応募した企業に就労した労働者の中から、定められた割合で離職者を選定し、再び労働市場内で求職活動を開始する。

労働市場に投入された労働者と離職した労働者は、応募する企業を探し情報収集を行うため、1日が経過してから企業に応募するものとした。

第2節 企業エージェント

非正規雇用の求人を募集している飲食企業を企業エージェントと置く。企業エージェントの種類はレベルによってのみ分類され、すべての企業の求人枠数は1～4人分という設定にした。求人枠数の合計は、飲食業の有効求人倍率（3.6倍）をもとにした12,960とし、企業数は9,237とした。企業エージェントの各レベルごとの割合は、飲食店.COM「飲食店経営に関するレポート」を参考にし、表3-2は各企業エージェントの割合とレベルを示したものである。

表 3-2 企業のレベル別求人枠数

レベル	1	2	3	4	5	6	合計
割合	20%	30%	20%	15%	10%	5%	100%
求人枠数	2,592	3,888	2,592	1,944	1,296	648	12,960

企業エージェントは企業の魅力を示す値として労働者エージェントと同様1～6のレベルを持ち、各レベルの企業数は飲食企業が求人に募集する賃金ごとの募集数を想定し割り当てた。モデルの設定として、非正規雇用の労働市場において、企業は労働力を早急に確保することを望み、企業は労働者を簡単に除外することができるため、質の高い労働者を求めるよりも質の低すぎる労働者以外を採用する傾向にある。したがって、企業が採用したいと考える労働者のレベルは、その企業のレベルより一つレベルを落としたレベルと同等かそれ以上という設定にした。

採否判定をするまでの日数は、労働者が企業に応募してから、企業が学生に採否の結果を通知するまでの日数である。この値は、正規分布 $N(\mu=4, \sigma=1)$ による乱数値が設定され、応募のたびに变化するものとした。

各企業の求人枠数は、生活衛生関係営業経営実態調査の従業者規模別施設数の構成割合を参考にした。表3-3は、日本の飲食店に雇用されている従業者の内訳を示したものである[8]。

表 3-3 従業者規模別の飲食店舗数

従業者規模	1人	2人	3人	4人	5～9人	10～19人	20人以上	不詳	合計
割合	7.1%	20.8%	11.5%	9.6%	30.3%	11.5%	7.1%	2.1%	100%

本分析では、上記をもとに企業の求人枠数と求人枠ごとの企業数を設定した。従業者規模が大きい企業ほど求人枠数が多いと仮定し、従業者規模1～4人の企業の求人枠数は1、従業者規模5～9人の企業の求人枠数は2、従業者規模10～19人の企業の求人枠数は

3、従業者規模 20 人以上の企業の求人枠数は 4 とし、それぞれの企業の割合を表 3-4 のように設定した。求人枠数 1 人分の企業は 3,685 社、求人枠数 2 人分の企業は 2,273 社、求人枠数 3 人分の企業は 861 社、求人枠数 4 人分の企業は 537 社の割合で存在し、全ての企業はそのどれかにランダムで振り分けられる。

表 3-4 従業者規模別企業数

規模	1~4 人	5~9 人	10~19 人	20 人以上	合計
割合	50.1%	30.9%	11.7%	7.3%	100%
求人枠	1	2	3	4	(平均 1.76 人)
求人枠数割合	28.4%	35.1%	19.9%	16.6%	100%
求人枠数	3,685	4,546	2,582	2,148	12,960
企業数	3,685	2,273	861	537	7,355

労働市場全体の最適化と企業の収益性を把握するために、企業エージェントごとに利益構造（売上とコスト）の概念を設ける。具体的には企業エージェントごとに客席数と客単価の内部データを設定する。

各企業エージェントの売上を決定する情報は客席数と客単価であり、1 日あたりの売上を以下の式で表す。

$$1 \text{ 日あたり売上} = \text{客席数} \times \text{満席率} \times \text{稼働回数} \times \text{客単価}$$

飲食企業の客席数は従業者規模が大きいほど多いと仮定し、客席数のよって必要人数と初期人数と初期充足率を決定する。満席率は顧客からの評価を表しており、0.5, 0.6, 0.7, 0.8 の中からランダムに振り分けられる。稼働回数は店舗の回転率の変動により変化し限界稼働回数に充足率を乗じた値である。客単価はランダムに振り分けられ、それが高い店舗ほど限界稼働数が低くなると仮定する。充足率は労働者数を必要人数で割った値である。これをまとめたものを表 3-5 と表 3-6 に示す。

表 3-5 従業者規模別客席数

規模	1~4 人	5~9 人	10~19 人	20 人以上
客席数	20	50	100	150
必要人数	2	6	12	20
初期人数	1	4	9	16
初期充足率	1/2	2/3	3/4	4/5

表 3-6 客単価対限界稼働回数

客単価	1000 円	2000 円	3000 円	4000 円	5000 円
-----	--------	--------	--------	--------	--------

限界稼働回数	12	8	6	4.8	4
--------	----	---	---	-----	---

各企業エージェントのコストは、企業レベルによる賃金と労働者数の増減によって定められる。1日あたりのコストを以下の式で表す。

$$1 \text{ 日あたりコスト} = \text{客席数} \times \text{満席率} \times \text{稼働回数} \times \text{食材費} + \text{労働者数} \times \text{賃金} \times \text{時間 (6h)} \\ + \text{求人コスト}$$

各企業の食材費は客単価に比例するものと仮定し、客単価の30%とする。時給は飲食店.COM「飲食店経営に関するレポート」の賃金割合構成を参考にし、表3-7の通りである。

表 3-7 レベル別時給

レベル	1	2	3	4	5	6
時給	980 円	1080 円	1180 円	1280 円	1380 円	1480 円

求人コストは、実際の求人広告の相場が一週間あたり2万円であるため、1日に求人枠1枠につき求人コスト3000円がかかるものとする。エージェントベースモデルによって挙動の変化を分析するため、家賃や水道光熱費などの固定費はコストに含まないものとする。

第3節 モデルの妥当性

モデルの妥当性を検証するために、前述した労働市場を基盤に他業種を含めた非正規雇用の主要な労働市場全体でのジョブマッチングを行った。尚、実際の統計結果と比較し検証するために、すでにデータがある2018年の労働市場をモデル化した。他業種を含めた労働市場の場合は、労働者エージェントと企業エージェントは互いに魅力度の幅が広がるため、1～6のレベルを1～8のレベルの幅に拡張し、各労働者の魅力度を再定義する。以下の表は労働者エージェントのレベルを再定義したものであり、元のモデル内でも全非正規雇用の割合を使用していたため、各労働者エージェント割合は不変とする。

表 3-8 全業種における各労働者エージェントのレベル

種類	フリーター	学生	主婦	外国人
割合	3割 (30%)	4割 (37%)	2割 (24%)	1割 (9%)
レベル	2-8	2-7	2-7	1-4

非正規雇用労働市場の主要な業種は、販売、飲食、製造・物流・清掃、事務、専門職の5つに分けられる。各企業エージェントのレベルと企業ごとの割合は株式会社リクルートジョブズの「2019年12月度アルバイト・パート募集時平均時給調査」を元に設定する。そのレベルを元に各業種のレベル別企業数割合を正規分布 $N(\mu = \text{レベル}, \sigma = 1)$ によって割り当てる。ただし、企業エージェントには最低賃金が設定されているため、レベルつまり平均賃金が最低賃金に近いほど、対数正規分布に近似するものとする。今回のように企業エージェント側に多様な業種がある場合、業務内容や職場環境などが大きく異なるため、それが労働者にとっての企業の魅力度に大きく影響すると考えられる。したがって、上で挙げた業種において就労経験がある10人を対象に賃金以外の魅力度のアンケート調査を実施し、それを元に相対的に点数が高い業種は企業数割合の初期レベルに1～2点を加算する。これをまとめたものを表3-9と図3.2に示す。

表 3-9 全業種における各企業エージェントのレベル

業種	販売	飲食	製造・物流・ 清掃	事務	専門職
割合	19%	26%	18%	15%	22%
レベル	1	1	2	2	3
加算点	1	0	0	2	0

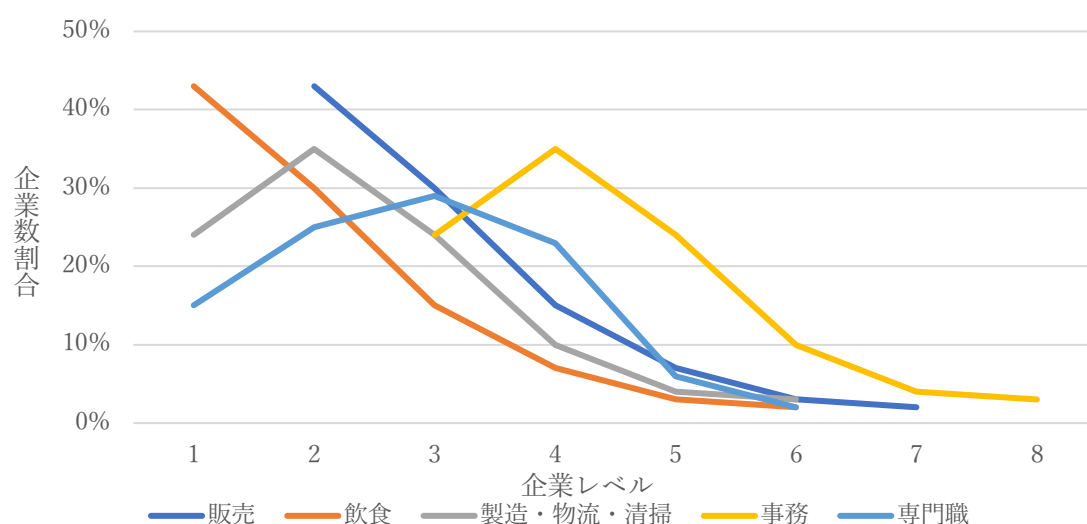


図 3.2 業種ごとのレベル別企業数割合

労働者エージェントの人数は3,600人であり、フリーター1,080人、学生1,440人、主婦720人、外国人360人とする。企業エージェントの求人枠数は非正規雇用労働市場の有効求人倍率(1.92倍)をもとに6,912人分とし、各業種に指定の割合で振り分けられ、各業種のレベル別企業数割合によってレベルが決まる。非正規雇用者の主要な業種の

有効求人倍率は、厚生労働省が出している「一般職業紹介状況」の参考統計表から、非正規雇用者（パート）のみの値を用いて算出した値であり、検証の比較対象である各業種の有効求人倍率も同様である。有効求人倍率は有効求人件数を有効求職者数で割って求めるため、モデルにおいては求人枠数を応募人数で割って有効求人倍率の理論値を計算した。表 3-10 は実際の統計データから 2018 年の有効求人倍率（非正規雇用）を業種ごとに算出した値とモデルのシミュレーションによって得られた有効求人倍率の理論値である。販売の有効求人倍率が相対的に低く計算されているが、全体的な大小関係はほぼ一致しており、値も似ていることから、このモデルの妥当性を確認できた。

表 3-10 実際の算出値とモデルによる理論値(有効求人倍率)

	2018 年度の有効求人倍率	モデルによる有効求人倍率
販売	3.02	2.43
飲食	4.53	4.92
製造・物流・清掃	2.80	3.15
事務	0.61	0.78
専門	2.34	2.69

第 4 章 分析結果

飲食業界の現状を表す基礎的な労働市場モデルとして、労働者エージェント 3,600 人分、企業エージェント 7,355 社分を設置しシミュレーション分析を行った。シミュレーションを開始すると、労働者の 1/360 が 1 日ごとに労働市場に投入され、求職活動を開始するものとした。

第 1 節 シナリオ分析

本分析では、シナリオごとにモデルを構築し、シミュレーションを行う。比較対象となるシナリオは 4 つである。シナリオ 1 は飲食業界の労働市場の現状を表したもので、求人倍率を 3.6 倍にした。シナリオ 2 は外部からの影響がないまま人手不足が解消した場合を表現しており、全労働者エージェントを増やし、有効求人倍率を 1.1 倍に設定した。シナリオ 3 は外国人労働者の増加により人手不足が緩和された状態を表しており、外国人労働者エージェントを増やし、有効求人倍率を 2.4 倍に設定した。シナリオ 4 は外国人労働者がさらに増加した状態であり、全体に占める外国人労働者の割合が相対的に大きい

(72%) と、外国人労働者の魅力度は全体の相対的魅力度に類似するため、外国人労働者のレベルの幅を（1～4）に修正し、有効求人倍率を 1.1 倍に設定した。実行結果は以下の表 4-1 の通りである。

まず、現在の飲食業界の労働市場（シナリオ1）の状態を確認する。求人倍率を3.6倍に設定しているため、労働者エージェントの就労率はほぼ97.4%と高い。一方で求人充足率は27.2%と低く、企業の殆どが労働者を確保できていない結果となった。

シナリオ1とシナリオ2を比較すると、求人倍率の調整により企業の求人充足率は上がったが、労働者の就労率は大きく下がる結果となった。求人倍率の調整を行なった同倍率のシナリオ2とシナリオ4を比較しても、時系列に沿って見てみると、シナリオ4は最後まで増加傾向があるのに対して、シナリオ2は時間=290ごろから減少傾向にあり、ミスマッチを起こしているのがわかる。就労率・求人充足率ともシナリオ4が優れている。これは飲食企業のレベル（賃金）別割合が低い方へ偏っているため、この労働市場ではレベルが低い企業への就労を望む労働者の割合が多い方が、全体最適化しやすいということを表している。したがって、労働者側と企業側の二つの視点から見ても、シナリオ4が最も全体最適化されたモデルといえる。

表 4-1 シナリオの実行結果

シナリオ	シナリオ 1	シナリオ 2	シナリオ 3	シナリオ 4
労働者エージェント数	3,600 人	11,780 人	5,460 人	11,780 人
企業エージェント数	7,355	7,355	7,355	7,355
求人枠数	12,960 人分	12,960 人分	12,960 人分	12,960 人分
求人倍率	3.6 倍	1.1 倍	2.4 倍	1.1 倍
外国人割合	9%	9%	40%	72%
結果				
就労数	3,505 人	10,026 人	5,251 人	11,125 人
就労率	97.40%	85.10%	96.20%	94.40%
求人充足率	27.20%	78.00%	40.80%	86.50%

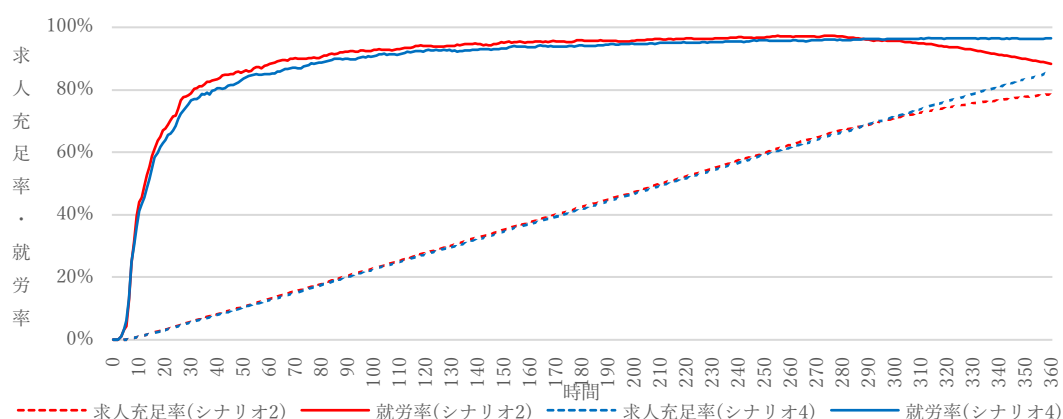


図 4.1 時系列充足・就労率

次に、シナリオ毎の特徴を明らかにするために、労働者がどの企業に就労したかを労働者のレベル別、企業のレベル別に集計した。図 4.1 は、各労働者のレベル別企業への就労数を表したものである。縦軸、横軸はそれぞれ企業のレベルと労働者のレベルを 1～6 まで示し、円の大きさは就労数の割合を表している。

シナリオ 2 を見ると、特にレベル 3 以上の労働者が自身よりもレベルが低い企業に就労している割合が多い。逆に言えば、レベル 2～4 の企業は自身のレベルよりもレベルが高く。外国人労働者が増加する過程をシナリオ 1、3、4 の順に仮定すると、外国人労働者の割合が増えるほど、レベルが高い企業の方が就労数の割合が下がる傾向にある。

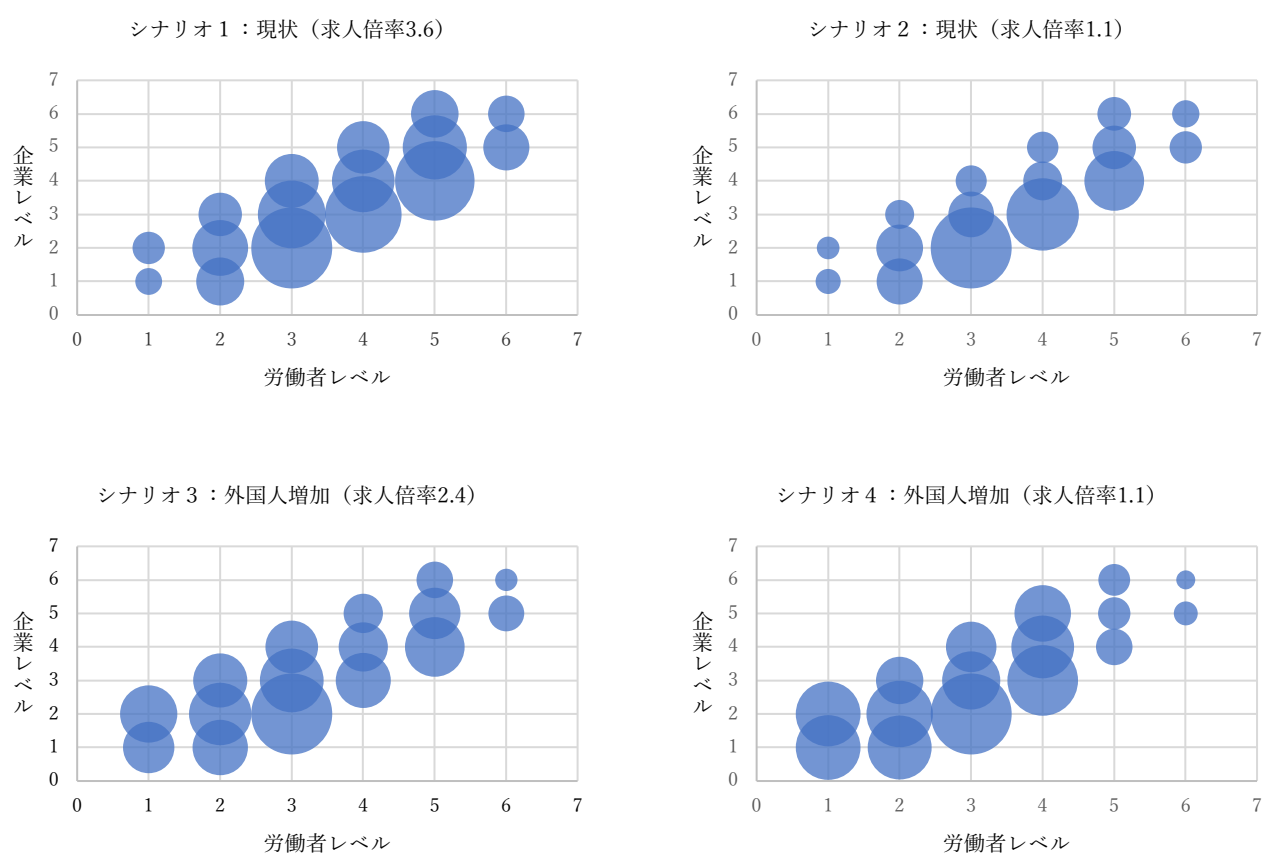


図 4.2 労働者と企業のレベル別マッチング状況

労働者の種類別就労状況の結果が図 4.2 と図 4.3 である。シナリオ 2 を見ると、現在の労働者種類別割合のまま求人倍率が落ち着くと、外国人以外の労働者の非就労比率が上がる事がわかる。現在の労働者種類別割合のまま求人倍率が落ち着くということは、日本の現在の人口動態から各労働者が全体的に増加することは考えにくいため、人手不足倒産により企業が淘汰され、人手不足が解消される場合が予想される。つまり、今の人手不足の状態が続き、企業が淘汰されれば、飲食業界において日本人側の失業率の方が高くなるという結果になった。

一方、外国人労働者の割合が増加し、求人倍率が落ち着く場合は、外国人労働者の就労数が大きくなるだけで、外国人以外の労働者の就労数はほぼ変わらない。これは外国人労働者の投入により、企業の倒産を防ぐことで、日本人側の雇用も守られることが示唆される。

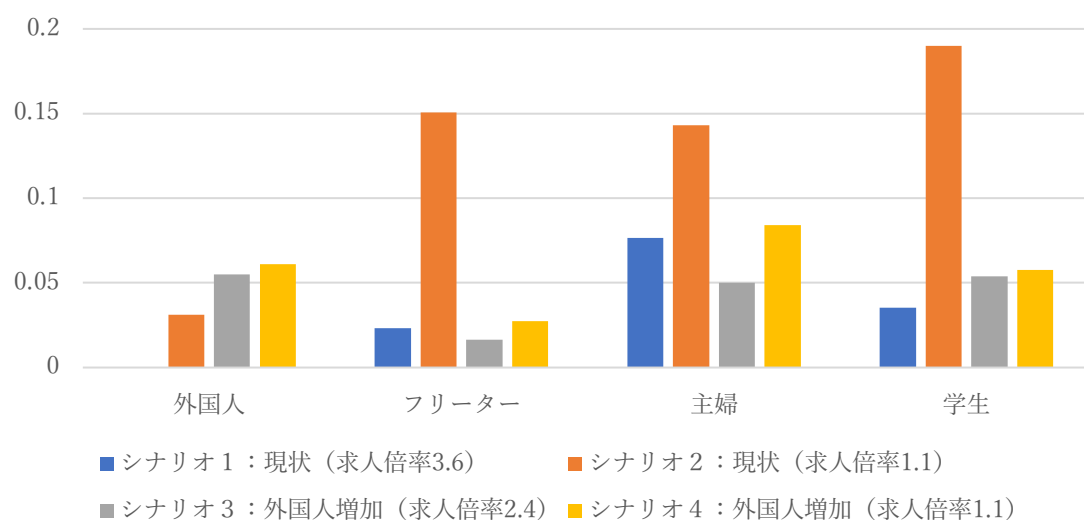


図 4.3 労働者種類別非就労率

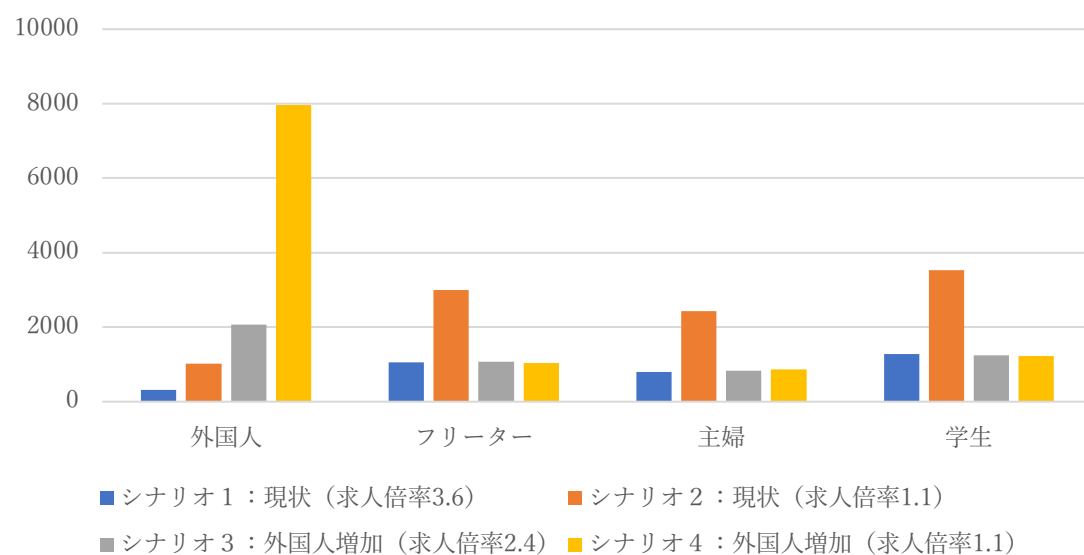


図 4.4 労働者種類別就労数

企業のレベル別就労状況から、将来の環境変化に対応した戦略提案を行う。シナリオ 1 を現在の飲食業界の状況とするならば、

シナリオ1からシナリオ2へ変化する場合（労働者種類別割合不変）：これが考えられるのは、人手不足に耐えられず、多くの飲食企業が淘汰され、求人倍率が落ち着いた時である。シナリオ2の企業レベル別求人充足率から、労働者確保のためには企業レベルを相対的に高く設定することが有効である。したがって、企業がとるべき戦略は他社よりも相対的に高い賃金を設定することである。

シナリオ1からシナリオ4へ変化する場合（外国人労働者割合増加）：これには政府の協力も必要であり、飲食企業の労働者の多くの割合を占めることになる。シナリオ4の企業レベル別求人充足率から、労働者確保のためには企業レベルを高く設定することが必要はない。したがって企業がとるべき戦略は外国人労働者が働く環境を整え、教育により企業内部で彼らのレベルを上げることである。

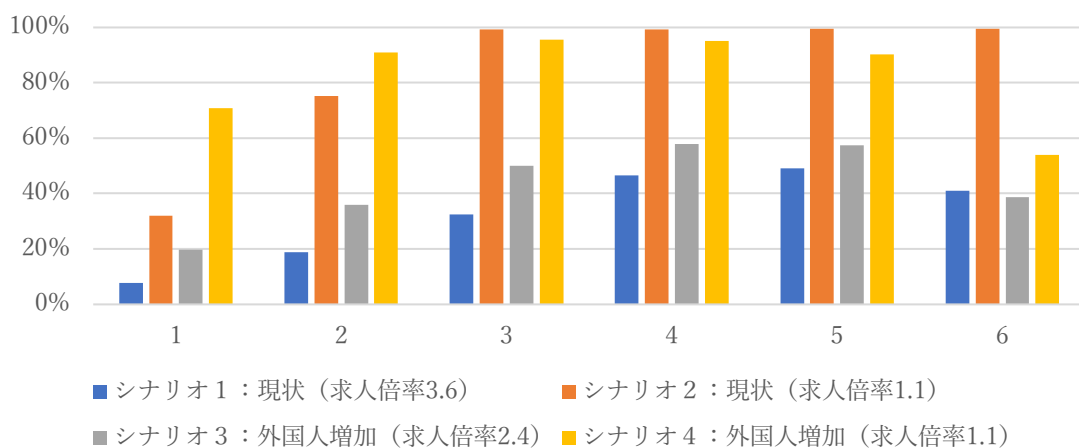


図 4.5 企業レベル別求人充足率

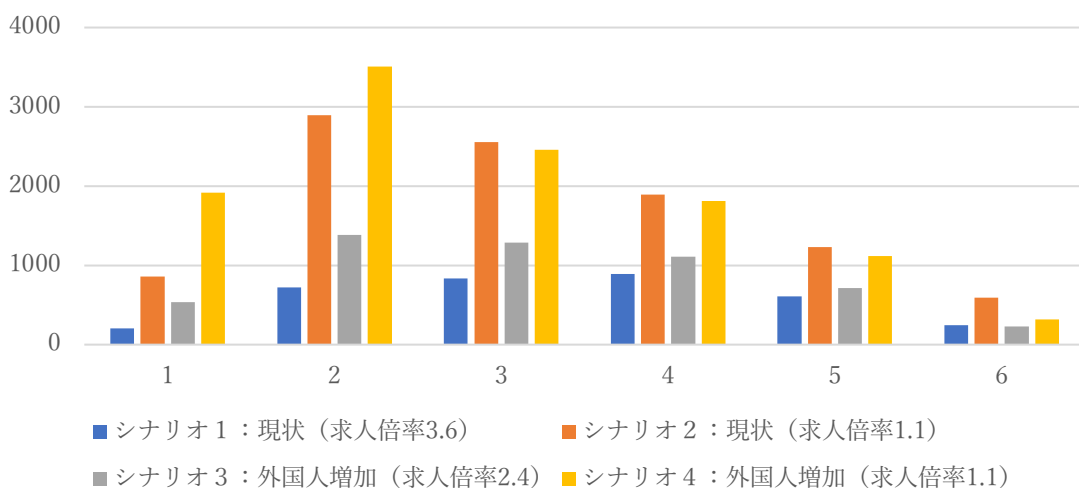


図 4.6 企業レベル別就労数

第2節 利益構造分析

本節では、シミュレーションにより得られた労働市場内部の利益構造を分析し、飲食業界の労働市場のメカニズムを解明したい。以下の表はシナリオ1、シナリオ2、シナリオ4（以降それぞれ現状シナリオ、現状維持シナリオ、外国人増加シナリオとする）の各レベル企業の平均利益と利益成長率を全体と企業特性ごとに示したものである。

全シナリオに共通して言えることは、平均利益も利益成長率もそれぞれ最適なレベルは異なるにも関わらず、平均利益率はレベル1の企業が最も高いということである。飲食業界において、経営者が賃金の意思決定を行う際に、将来の利益や利益成長率などの不確定要素よりも現実の利益率を重要視する可能性は高い。この結果は、飲食業界の低賃金化を招く要因であることが示唆される。

現状シナリオの利益を見ると、レベル4の企業が最適な戦略ではあるが、多レベル企業との差はさほどなく、現状維持シナリオや外国人増加シナリオに比べて平均利益も利益成長率も相対的に低いことがわかる。これは人手不足が企業の利益に悪影響を及ぼしていることを表しており、飲食企業において重大な課題であることが確認できる。

将来起こりうる現状維持シナリオと外国人増加シナリオを比較すると、現状維持シナリオは相対的に高いレベルを設定した企業の利益が多く、成長率も高い傾向にあり、一方外国人増加シナリオにそのような傾向は見られず、レベル1とレベル6の企業以外は全体的に高利益である。前述した「労働者種類別割合不変の場合は他社よりも相対的に高い賃金を設定し、外国人労働者割合増加の場合は賃金による競争よりも外国人労働者が働く環境を整え、教育により企業内部で彼らのレベルを上げる」という飲食企業への戦略提案の妥当性を裏付ける結果となった。

労働市場モデル内で設定した企業の特徴には席数、満席率、客単価があり、席数は飲食店の規模、満席率は飲食店の人気度、客単価は飲食店の客層を表している。シミュレーションの結果から、各シナリオ上の特性ごとに最適なレベルを計算することができたが、精度の低さが課題である。例えば現状維持シナリオのような明らかに相対的に高いレベルの企業が有利な状況で、低いレベルが最適な値として計算される場合がある。内部で蓄積されたデータを確認したところ、ある特性の企業の中に満席率が高い企業の偏りがあるなどの問題が発見された。これは特性を細かく分けるあまり分類された特性のサンプル数が少なくなったことが原因である。現在私が使用しているソフトではこれ以上のサンプル数を設定することができないが、技術の進歩とともにさらに大量のデータが分析可能なソフトの登場によりこの課題は解決するであろう。

表 4-2 全レベル別企業の利益(現状シナリオ)

	レベル1	レベル2	レベル3	レベル4	レベル5	レベル6	総計
平均利益	¥223,951	¥229,058	¥238,943	¥242,388	¥244,643	¥224,884	¥233,259

利益成長率	111%	120%	135%	140%	137%	136%	127%
平均利益率	61.7%	61.3%	61.3%	60.6%	60.3%	58.9%	61.1%

表 4-3 特性ごとのレベル別企業の利益(現状シナリオ)

席数	平均利益	利益成長率	満席率	平均利益	利益成長率	客単価	平均利益	利益成長率
20	¥81,582	141%	0.5	¥169,395	127%	1000	¥160,752	127%
レベル 1	¥75,281	118%	レベル 1	¥159,408	110%	レベル 1	¥172,957	107%
レベル 2	¥78,385	133%	レベル 2	¥158,640	121%	レベル 2	¥165,710	120%
レベル 3	¥85,786	153%	レベル 3	¥173,858	136%	レベル 3	¥149,291	139%
レベル 4	¥86,660	160%	レベル 4	¥201,521	138%	レベル 4	¥162,890	143%
レベル 5	¥86,476	157%	レベル 5	¥168,214	142%	レベル 5	¥113,313	150%
レベル 6	¥84,771	155%	レベル 6	¥160,959	131%	レベル 6	¥178,059	138%
50	¥233,761	114%	0.6	¥215,732	130%	2000	¥218,828	124%
レベル 1	¥225,186	105%	レベル 1	¥215,266	111%	レベル 1	¥171,277	113%
レベル 2	¥229,087	109%	レベル 2	¥208,025	123%	レベル 2	¥204,384	117%
レベル 3	¥241,204	116%	レベル 3	¥196,938	146%	レベル 3	¥264,816	132%
レベル 4	¥244,420	125%	レベル 4	¥235,828	142%	レベル 4	¥240,916	136%
レベル 5	¥240,840	125%	レベル 5	¥245,820	135%	レベル 5	¥260,729	135%
レベル 6	¥221,101	120%	レベル 6	¥197,325	140%	レベル 6	¥178,740	129%
100	¥514,879	108%	0.7	¥251,268	126%	3000	¥257,542	127%
レベル 1	¥505,894	102%	レベル 1	¥261,885	112%	レベル 1	¥279,496	113%
レベル 2	¥508,864	106%	レベル 2	¥252,178	116%	レベル 2	¥223,260	121%
レベル 3	¥547,359	109%	レベル 3	¥285,997	135%	レベル 3	¥247,401	134%
レベル 4	¥487,182	117%	レベル 4	¥201,457	148%	レベル 4	¥293,012	136%
レベル 5	¥547,409	112%	レベル 5	¥236,859	136%	レベル 5	¥268,221	137%
レベル 6	¥480,847	112%	レベル 6	¥214,296	142%	レベル 6	¥252,190	143%
150	¥835,432	104%	0.8	¥301,259	123%	4000	¥255,912	130%
レベル 1	¥801,794	101%	レベル 1	¥258,608	111%	レベル 1	¥203,150	114%
レベル 2	¥860,813	103%	レベル 2	¥302,782	122%	レベル 2	¥282,184	121%
レベル 3	¥820,416	105%	レベル 3	¥293,537	124%	レベル 3	¥278,857	134%
レベル 4	¥824,526	106%	レベル 4	¥323,529	132%	レベル 4	¥256,352	146%

レベル 5	¥938,260	107%	レベル 5	¥332,375	137%	レベル 5	¥199,849	142%
レベル 6	¥788,426	101%	レベル 6	¥367,993	128%	レベル 6	¥293,429	138%
総計	¥233,259	127%	総計	¥233,259	127%	5000	¥278,250	125%
						レベル 1	¥293,487	108%
						レベル 2	¥280,939	122%
						レベル 3	¥237,356	135%
						レベル 4	¥265,828	137%
						レベル 5	¥356,923	125%
						レベル 6	¥225,094	129%
						総計	¥233,259	127%

表 4-4 全レベル別企業の利益(現状維持シナリオ)

	レベル 1	レベル 2	レベル 3	レベル 4	レベル 5	レベル 6	総計
平均利益	¥235,612	¥258,136	¥280,618	¥288,207	¥293,916	¥267,098	¥266,515
利益成長率	138%	170%	180%	177%	176%	179%	168%
平均利益率	61.8%	61.7%	61.7%	61.0%	60.7%	59.3%	61.4%

表 4-5 特性ごとのレベル別企業の利益(現状維持シナリオ)

席数	平均利益	利益成長率	満席率	平均利益	利益成長率	客単価	平均利益	利益成長率
20	¥106,498	198%	0.5	¥195,412	170%	1000	¥182,753	167%
レベル 1	¥87,733	161%	レベル 1	¥167,336	136%	レベル 1	¥179,444	133%
レベル 2	¥102,281	203%	レベル 2	¥181,668	175%	レベル 2	¥187,096	167%
レベル 3	¥114,861	208%	レベル 3	¥208,959	182%	レベル 3	¥178,941	186%
レベル 4	¥118,079	210%	レベル 4	¥239,189	178%	レベル 4	¥192,557	180%
レベル 5	¥120,325	210%	レベル 5	¥201,816	180%	レベル 5	¥141,394	187%
レベル 6	¥112,881	210%	レベル 6	¥189,118	177%	レベル 6	¥206,957	174%
50	¥272,289	144%	0.6	¥247,091	168%	2000	¥251,151	171%
レベル 1	¥236,559	119%	レベル 1	¥225,946	138%	レベル 1	¥182,701	145%
レベル 2	¥264,407	145%	レベル 2	¥234,373	170%	レベル 2	¥232,527	178%
レベル 3	¥291,823	154%	レベル 3	¥233,879	183%	レベル 3	¥307,207	174%
レベル 4	¥298,873	154%	レベル 4	¥280,087	177%	レベル 4	¥280,807	181%
レベル 5	¥293,294	154%	レベル 5	¥294,926	171%	レベル 5	¥313,960	168%

レベル 6	¥264,741	155%	レベル 6	¥237,738	179%	レベル 6	¥221,877	188%
100	¥561,851	127%	0.7	¥285,678	166%	3000	¥295,028	166%
レベル 1	¥516,063	107%	レベル 1	¥274,365	135%	レベル 1	¥293,967	137%
レベル 2	¥545,183	126%	レベル 2	¥283,162	166%	レベル 2	¥250,732	172%
レベル 3	¥616,004	135%	レベル 3	¥327,995	177%	レベル 3	¥295,960	178%
レベル 4	¥547,638	136%	レベル 4	¥247,380	184%	レベル 4	¥350,766	171%
レベル 5	¥627,267	136%	レベル 5	¥294,705	177%	レベル 5	¥320,359	170%
レベル 6	¥549,063	135%	レベル 6	¥255,995	184%	レベル 6	¥301,271	178%
150	¥883,036	120%	0.8	¥343,076	167%	4000	¥290,370	168%
レベル 1	¥811,546	105%	レベル 1	¥274,461	142%	レベル 1	¥211,902	137%
レベル 2	¥887,633	116%	レベル 2	¥339,240	169%	レベル 2	¥318,129	167%
レベル 3	¥881,312	127%	レベル 3	¥345,717	178%	レベル 3	¥317,769	179%
レベル 4	¥902,837	127%	レベル 4	¥379,026	173%	レベル 4	¥303,439	179%
レベル 5	¥1,026,759	126%	レベル 5	¥387,616	176%	レベル 5	¥246,569	184%
レベル 6	¥878,455	125%	レベル 6	¥432,636	173%	レベル 6	¥327,387	173%
総計	¥266,515	168%	総計	¥266,515	168%	5000	¥319,023	166%
						レベル 1	¥310,764	136%
						レベル 2	¥314,339	166%
						レベル 3	¥285,487	182%
						レベル 4	¥321,186	176%
						レベル 5	¥419,092	170%
						レベル 6	¥287,675	182%
						総計	¥266,515	168%

表 4-6 全レベル別企業の利益(外国人増加シナリオ)

	レベル 1	レベル 2	レベル 3	レベル 4	レベル 5	レベル 6	総計
平均利益	¥256,552	¥268,606	¥278,855	¥275,984	¥266,822	¥226,248	¥266,922
利益成長率	167%	175%	180%	175%	169%	150%	172%
平均利益率	62.3%	61.9%	61.7%	60.8%	60.2%	58.6%	61.4%

表 4-7 特性ごとのレベル別企業の利益(外国人増加シナリオ)

席数	平均利益	利益成長率	席数	平均利益	利益成長率	客単価	平均利益	利益成長率
20	¥107,415	203%	20	¥107,415	203%	1000	¥185,889	172%

レベル 1	¥102,606	197%	レベル 1	¥102,606	197%	レベル 1	¥198,621	165%
レベル 2	¥106,719	206%	レベル 2	¥106,719	206%	レベル 2	¥195,666	172%
レベル 3	¥115,457	209%	レベル 3	¥115,457	209%	レベル 3	¥180,357	186%
レベル 4	¥111,910	206%	レベル 4	¥111,910	206%	レベル 4	¥185,096	177%
レベル 5	¥105,621	201%	レベル 5	¥105,621	201%	レベル 5	¥126,080	171%
レベル 6	¥87,864	176%	レベル 6	¥87,864	176%	レベル 6	¥183,268	144%
50	¥273,816	149%	50	¥273,816	149%	2000	¥250,243	176%
レベル 1	¥264,132	145%	レベル 1	¥264,132	145%	レベル 1	¥203,588	176%
レベル 2	¥278,332	151%	レベル 2	¥278,332	151%	レベル 2	¥239,078	181%
レベル 3	¥289,306	153%	レベル 3	¥289,306	153%	レベル 3	¥304,953	173%
レベル 4	¥285,572	153%	レベル 4	¥285,572	153%	レベル 4	¥265,801	178%
レベル 5	¥262,547	149%	レベル 5	¥262,547	149%	レベル 5	¥280,348	162%
レベル 6	¥221,373	131%	レベル 6	¥221,373	131%	レベル 6	¥187,744	165%
100	¥558,206	130%	100	¥558,206	130%	3000	¥293,168	170%
レベル 1	¥544,227	126%	レベル 1	¥544,227	126%	レベル 1	¥312,753	160%
レベル 2	¥561,185	132%	レベル 2	¥561,185	132%	レベル 2	¥263,032	178%
レベル 3	¥605,820	134%	レベル 3	¥605,820	134%	レベル 3	¥291,345	179%
レベル 4	¥524,654	134%	レベル 4	¥524,654	134%	レベル 4	¥334,991	171%
レベル 5	¥583,200	131%	レベル 5	¥583,200	131%	レベル 5	¥292,375	167%
レベル 6	¥474,952	114%	レベル 6	¥474,952	114%	レベル 6	¥247,327	139%
150	¥882,055	121%	150	¥882,055	121%	4000	¥291,376	173%
レベル 1	¥835,589	114%	レベル 1	¥835,589	114%	レベル 1	¥234,476	173%
レベル 2	¥918,714	122%	レベル 2	¥918,714	122%	レベル 2	¥331,532	170%
レベル 3	¥878,158	126%	レベル 3	¥878,158	126%	レベル 3	¥320,096	180%
レベル 4	¥872,984	124%	レベル 4	¥872,984	124%	レベル 4	¥288,287	178%
レベル 5	¥957,511	120%	レベル 5	¥957,511	120%	レベル 5	¥220,148	178%
レベル 6	¥799,164	107%	レベル 6	¥799,164	107%	レベル 6	¥281,594	146%
総計	¥266,922	172%	総計	¥266,922	172%	5000	¥319,583	171%
						レベル 1	¥334,642	162%
						レベル 2	¥326,874	173%
						レベル 3	¥279,412	181%
						レベル 4	¥312,673	170%
						レベル 5	¥388,880	165%
						レベル 6	¥236,446	161%
						総計	¥266,922	172%

現状維持シナリオと外国人増加シナリオを包括的に比較するために飲食業界全体の利得について分析を行う。飲食業界の中で利得を得るものは企業と労働者であり、各々の視点で利得を計算し比較する。

企業側の利得について、全企業の利益の合計によって現状維持シナリオと外国人増加シナリオを比較する。図 4.7 を見ると企業エージェントの利益は同じように増加するが、後半で現状維持シナリオが減退したため、結果的に最後まで増加を続けた外国人増加シナリオが企業エージェントにより多く利益を出させた。

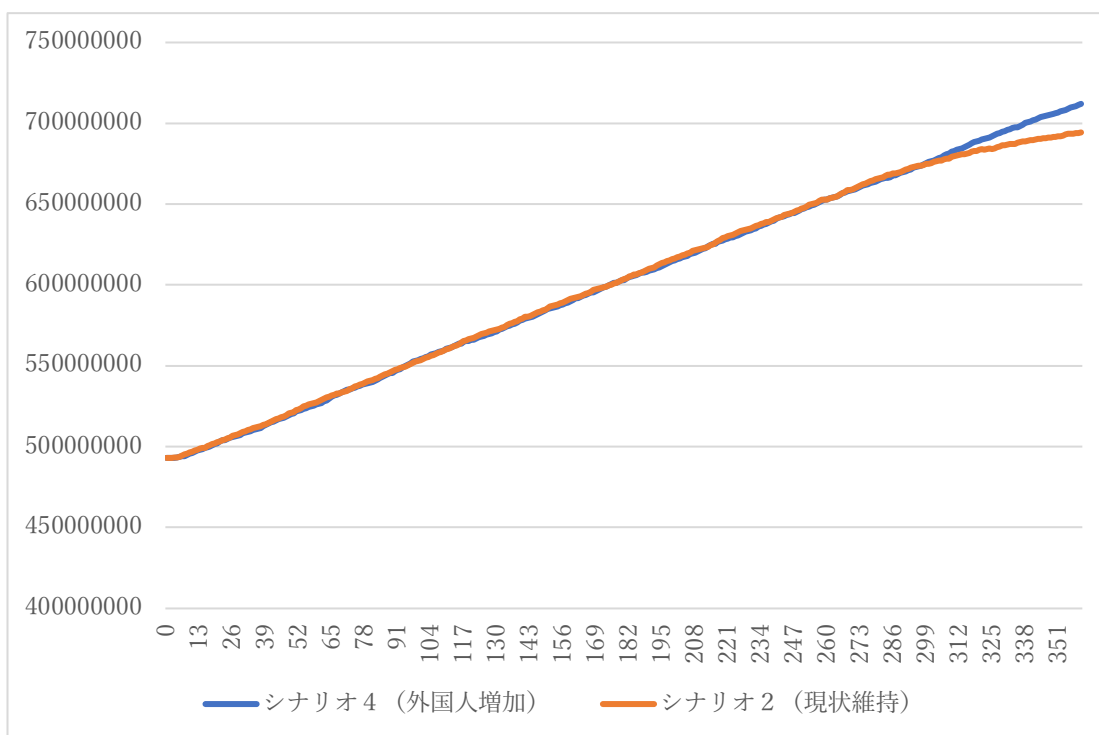


図 4.7 企業の利益合計推移

労働者側の利得について、全労働者の賃金の合計と平均賃金をそれぞれ図 4.8 と図 4.9 に示した。図 4.8 を見ると現状維持シナリオの方が全労働者エージェントの賃金合計の上昇率も高く、外国人増加シナリオよりも継続して高い値を保っていたが、290 日頃からその勢いを失くし、最終的に外国人増加シナリオ 4 に逆転された。したがって、外国人増加シナリオの方が労働者全体への利得が高い結果となった。

しかし、労働者エージェントの平均賃金に着目すると、現状維持シナリオは増加傾向にあるが、外国人増加シナリオは微減傾向にある。したがって外国人を増加させ続ければ、賃金の調整メカニズムは働かず、各労働者の賃金はいつまで経っても上がらないことが示唆される。

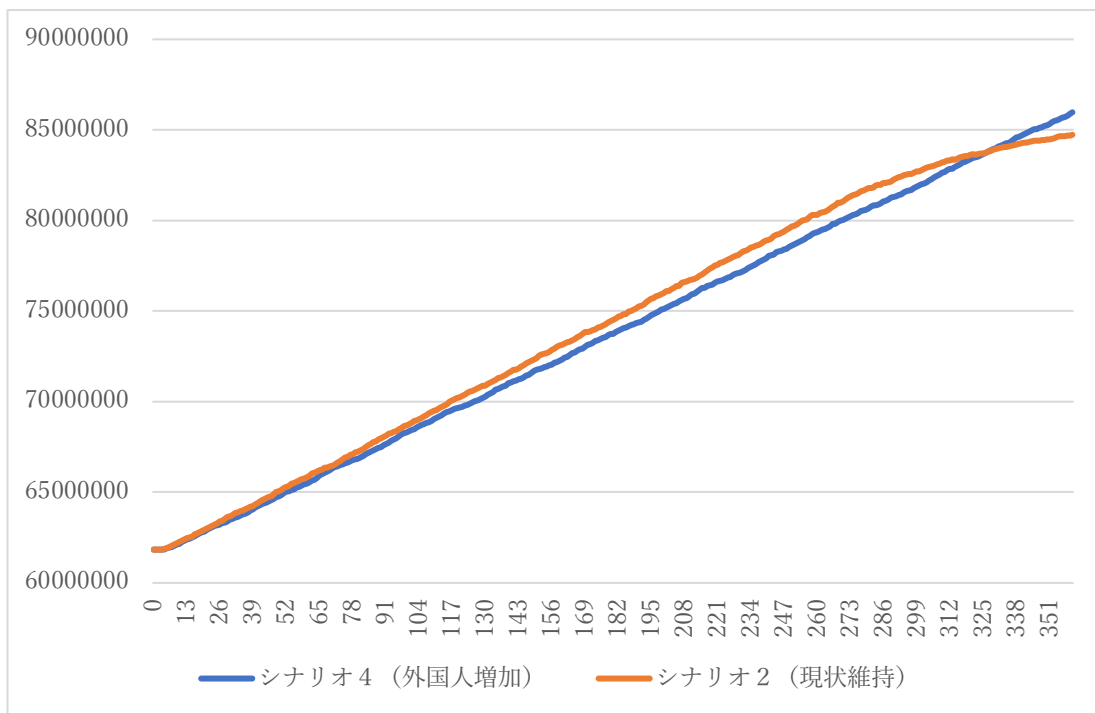


図 4.8 労働者の賃金合計の推移

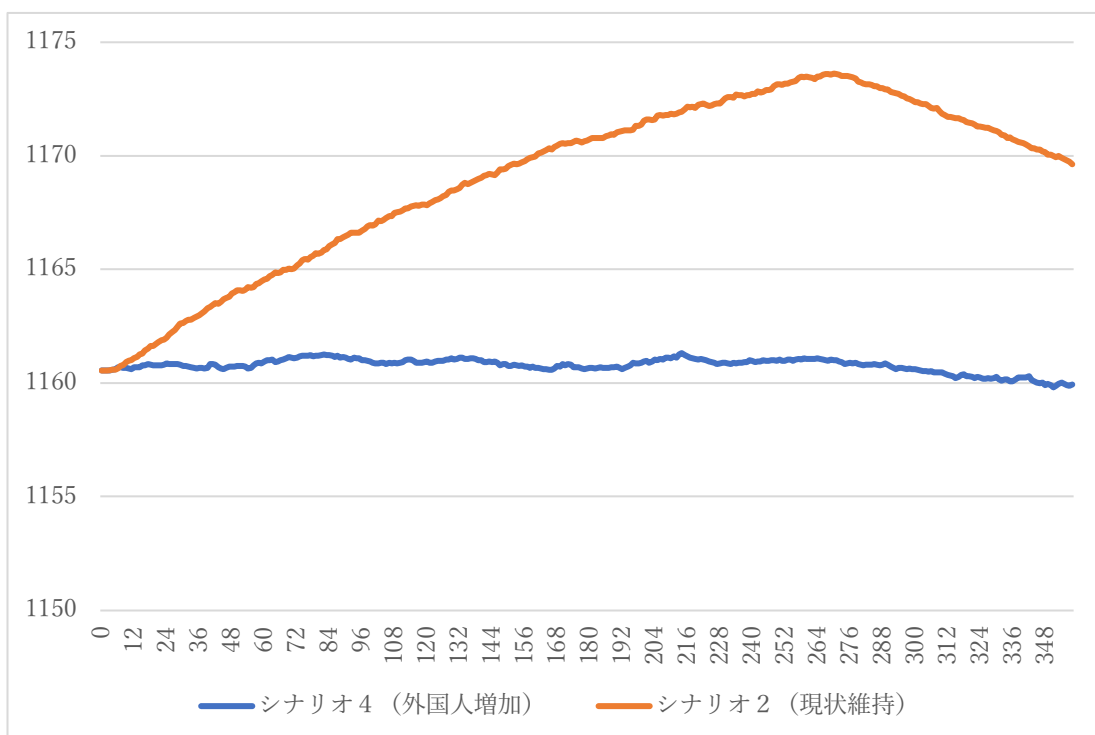


図 4.9 労働者の平均賃金推移

第5章 考察

求人充足率と就労率の二つの視点で、労働市場全体の最適化を考えた場合、外国人労働者の全体に占める割合を増やし、人手不足を解消すべきであるという結論に達した。

現状シナリオの分析から現在の飲食企業の労働市場の特徴として、労働者の高い就労率が挙げられるが、その一方で企業の低すぎる求人充足率も明らかとなった。外国人労働者の割合増加による人手不足解消は、現在の高就労率を保ったまま、企業の求人充足率も上げることが可能である。

しかし、外国人を増加させ続ければ労働者の賃金はいつまでも上がらない。政府の施策は企業の利益を確保または延命させるほかない。

すでに外国人労働者が飲食企業の労働者に占める割合は年々増加しているため、飲食企業の経営者は、外国人労働者を主戦力と位置づけ、彼らが求めるものは何か、彼らが働きやすい環境を作るためには何が必要かを真剣に考えるべきである。

第6章 まとめ

飲食企業の労働市場は、非正規雇用者によって支えられているという点で他産業とは異なる特徴を有しており、現在最も問題を抱えている労働市場の一つである。本研究では、この課題を解決するために、飲食企業の労働市場のモデル化を行い、シナリオ分析を通じ、その特徴を示すことができた。今後の課題としては、さらにデータ量を増やしたモデルの構築、地理的要素を含むモデル化、強化学習などがあげられる。

参考文献

- [1] 総務省：労働力調査 長期時系列データ，（2019），
<https://www.stat.go.jp/data/roudou/longtime/03roudou.html>
- [2] 総務省：平成 29 年就業構造基本調査，（2017），
<https://www.stat.go.jp/data/shugyou/2017/index.html>
- [3] 厚生労働省：外国人雇用状況，（2019），
https://www.mhlw.go.jp/stf/newpage_03337.html
- [4] 森敬子・倉橋節也：エージェントベースシミュレーションによる新規学卒者採用市場における学生行動分析，経営情報学会誌，（2013）
- [5] 倉橋節也，企業活動のモデル化とシミュレーション，計測と制御 第 55 巻 第 1 号 2016 年 1 月号，計測自動制御学会，（2016）

- [6] 帝国データバンク：人手不足に対する企業の動向調査, (2019),
<https://www.tdb.co.jp/report/watching/press/p190506.html>
- [7] 飲食店.COM：飲食店経営に関するレポート (2019),
<https://www.inshokuten.com/research/magazine/>
- [8] 株式会社リクルートジョブズ：2018 年 12 月度アルバイト・パート募集時平均時給調査 (2019), https://www.recruitjobs.co.jp/press/pr20190116_1065.html
- [9] 厚生労働省：一般職業紹介状況, (2019),
<https://www.mhlw.go.jp/toukei/list/114-1.html>
- [10] 厚生労働省：生活衛生関係営業経営実態調査, (2014),
https://www.mhlw.go.jp/stf/seisakunitsuite/bunya/kenkou_iryoku/kenkou/seikatsu-eisei/seikatsu-eisei22/index.html

Appendix

A.1 プログラムコード

```
# -*- Mode: Python; coding: utf-8-dos; -*-

import sys; sys.path.insert(0, u"C:\\Program Files\\Mathematical Systems Inc\\S-
Quattro Simulation System V5\\lib3")
from psim import *
from graph import graphsendclient as graphclient
import datetime
import hashlib
graphclient.GraphServer.createServer(uri='http://127.0.0.1:57121')

initialize()

setGlobalSeed(0)

# カスタムコード
import random

# シミュレーションパラメータの定義
class SimulationParam:
    def __init__(self,
                  inputDir = u"input\\default",
```

```

        outputDir = u"output\\default"):
    pass

# シミュレーターの定義
class Simulator (SimulatorBase):
    def __init__(self,
        param = SimulationParam(),
        inputDir = u"input\\default",
        outputDir = u"output\\default",
        showGraph = True,
        start = lambda self: None,
        warm = lambda self: 0.0,
        until = lambda self: 360.0,
        odesolver = lambda self: dopri54Solver(dt = 0.1, rtol = 0.001, atol
= 1e-06),
        dataDir = u"data"):
        SimulatorBase.__init__(self, param, inputDir, outputDir, showGraph, start,
warm, until, odesolver, dataDir)

# 資源の定義

# オブジェクトの生成
# リンクの生成
# 格子グラフ の定義

class LatticeGraph (LatticeGraphBase):
    def __init__(self, **keys):
        LatticeGraphBase.__init__(self, u"格子グラフ", width = 5, height = 5,
type = 8, directed = False, **keys)
    def initAfter(self, **keys):
        pass
    def initAttribute(self, agent):
        pass
    def setPosition(self, agent, pos):
        try:
            origPos = self._agentAttrs[agent]

```

```

        if isinstance(origPos, (list, tuple)):
            # エッジ上
            self.graph[origPos[0]][origPos[1]]["agents"].remove(agent)
        else:
            # ノード上
            self.graph.node[origPos]["agents"].remove(agent)
    except:
        pass
    self._agentAttrs[agent] = pos
    if isinstance(pos, (list, tuple)):
        # エッジ上
        self.graph[pos[0]][pos[1]]["agents"].append(agent)
    else:
        # ノード上
        self.graph.node[pos]["agents"].append(agent)
def getPosition(self, agent):
    return self._agentAttrs[agent]
def getRandomPosition(self):
    return next(self.gnodeOrEdge)
def draw(self, panel):
    screen = panel.screen
    for (u, v) in self.graph.edges_iter():
        (x0, y0) = self.layout[u]
        (x1, y1) = self.layout[v]
        screen.line(x0, y0, x1, y1, alpha = 0.3)
def drawAgents(self, panel, agents):
    screen = panel.screen
    for agent in agents:
        pos = agent.getPosition()
        if isinstance(pos, (list, tuple)):
            (u, v, d) = pos
            p0 = numpy.asarray(self.layout[u])
            p1 = numpy.asarray(self.layout[v])
            p = (1 - d) * p0 + d * p1
            screen.point(p[0], p[1],
                        size = agent.screenSize,

```

```

        color = agent.screenColor,
        marker = agent.screenMarker,
        alpha = agent.screenAlpha)
    else:
        (x, y) = self.layout[pos]
        screen.point(x, y,
                    size = agent.screenSize,
                    color = agent.screenColor,
                    marker = agent.screenMarker,
                    alpha = agent.screenAlpha)

    def custom(self):
        pass

self.eLatticeGraph = LatticeGraph(simulator = self)

self.addEnv(self.eLatticeGraph)
# 企業エージェント の定義

class SynchronousAgentSet2 (SynchronousAgentSetBase):
    def _defAgent(self):
        class _Agent (SynchronousAgentBase):
            def initAfter(self, *args, **keys):

                x = random.randint(0,4)
                y = random.randint(0,4)
                p = self.agentset.env.xy2node[(x,y)]
                self.setPosition(p)
                self.screenColor = 'r'
                self.screenSize = 30

                # type
                self.type = "company"

                # レベル
                self.level = next(self.agentset.levelGen)

```

```

# 企業規模
self.scale = next(self.agentset.scaleGen)

# 企業規模によって席数、必要人数、初期人数が変わる(常に
初期人数分はいるとする、求人枠は固定)
if self.scale == 1:
    self.frame = 1
    self.seats = 20
    self.memberNum = 1
elif self.scale == 2:
    self.frame = 2
    self.seats = 50
    self.memberNum = 4
elif self.scale == 3:
    self.frame = 3
    self.seats = 100
    self.memberNum = 9
elif self.scale == 4:
    self.frame = 4
    self.seats = 150
    self.memberNum = 16

#満席率
self.occupancy = next(self.agentset.occupancyGen)

#単価
self.price = next(self.agentset.priceGen)

# 食材原価
self.foodCost = self.price * 0.3

if self.price == 1000:
    self.turnNumMax = 12
elif self.price == 2000:
    self.turnNumMax = 8

```

```

elif self.price == 3000:
    self.turnNumMax = 6
elif self.price == 4000:
    self.turnNumMax = 4.8
elif self.price == 5000:
    self.turnNumMax = 4

# 応募中のエージェント
self.apply = []

# 採用中のエージェント
self.adopt = []

# 稼働回数
self.turnNum = self.turnNumMax * ((self.memberNum +
len(self.adopt)) / (self.memberNum + self.frame))

def step(self):
    for a in self.apply:
        if a.wait != 0:
            a.wait -= 1
        else:
            if self.level - 1 <= a.level: # 採用
                self.adopt.append(a)
                a.elapsedNum = 0
                a.state = "就職中" # 就職中
                self.apply.remove(a)
                a.wait = 0
            else: # 不採用
                a.state = "情報収集中"
                a.company = None
                a.elapsedNum = 0
                self.apply.remove(a)
                a.wait = 0

```

```
self.agentset.monitor2.observe(now(),self.agentid,self.level,len(self.adopt),len(self.app
ly),self.frame)
```

```
        # コスト(客席*満席率*稼働回数*食材原価+労働者数*賃金*時間
+求人コスト)
```

```
        self.turnNum = self.turnNumMax * ((self.memberNum +
len(self.adopt)) / (self.memberNum + self.frame))
```

```
        self.cost = self.seats * self.occupancy * self.turnNum *
self.foodCost + (self.memberNum + len(self.adopt)) * self.agentset.wage[self.level]*6
+ (self.frame-len(self.adopt))*3000
```

```
        # 売上
```

```
        self.sales = self.seats * self.occupancy * self.turnNum *
```

```
self.price
```

```
self.agentset.monitor6.observe(now(),self.agentid,self.sales,self.cost,self.seats,self.occ
upancy,self.turnNum,self.foodCost,(self.memberNum +
len(self.adopt)),self.agentset.wage[self.level]*6,(self.frame-
len(self.adopt))*3000,self.price)
```

```
def custom(self):
```

```
    pass
```

```
    return _Agent
```

```
def initAfter(self, **keys):
```

```
    self.interval = 1.0 # ステップ間隔
```

```
    self.freezeAddAgent = True # エージェントの追加を次ステップまで
```

凍結する

```
    self.freezeDelAgent = True # エージェントの削除を次ステップまで凍
```

結する

```
    """エージェント集合の作成後にこのメソッドが呼ばれる。
```

```
    エージェント集合固有の初期化を行う。"""
```

```
self.agentFreezeVars = [] # エージェントの凍結する変数リスト
```

```
self.levelGen = empiricalDistribution(weights = [[20, 1],  
                                                [30, 2],  
                                                [20, 3],  
                                                [15, 4],  
                                                [10, 5],  
                                                [5, 6]  
                                                ])
```

```
self.scaleGen = empiricalDistribution(weights = [[50.1, 1],  
                                                [30.9, 2],  
                                                [11.7, 3],  
                                                [7.3, 4]  
                                                ])
```

```
self.occupancyGen = empiricalDistribution(weights = [[1, 0.5],  
                                                    [1, 0.6],  
                                                    [1, 0.7],  
                                                    [1, 0.8]  
                                                    ])
```

```
self.priceGen = empiricalDistribution(weights = [[1, 1000],  
                                                [1, 2000],  
                                                [1, 3000],  
                                                [1, 4000],  
                                                [1, 5000]  
                                                ])
```

```
# 賃金
```

```
self.wage = {1:980,2:1080,3:1180,4:1280,5:1380,6:1480}
```

```
# エージェントの生成
```

```

n = 9237 # エージェントの生成数
#n = 360000
self.generateAgents(n, **keys)


# 内定率
self.adoptRate = 0


self.monitor = Monitor(["時刻", "充足率"], ["f", "f"], name="充足率")
self.simulator.addMonitor(self.monitor)


self.monitor2 = Monitor(["時刻", "エージェント ID", "レベル", "採用数", "
応募数", "採用率"], ["f", "i", "i", "i", "i", "i"], name="企業")
self.simulator.addMonitor(self.monitor2)


self.monitor3 = Monitor(["時刻", "レベル", "採用数", "応募数", "採用率"], ["f", "i", "i", "i", "i"], name="レベルごとのログ")
self.simulator.addMonitor(self.monitor3)


self.monitor4 = Monitor(["時刻", "労働者レベル", "労働者種別", "企業レベル", "就労数"], ["f", "i", "o", "i", "i"], name="就労ログ")
self.simulator.addMonitor(self.monitor4)


self.monitor5 = Monitor(["時刻", "就労している人数", "応募している人数", "投入されている人数"], ["f", "i", "i", "i"], name="内定ログ")
self.simulator.addMonitor(self.monitor5)


self.monitor6 = Monitor(["時刻", "エージェント ID", "売上", "コスト", "席数", "満席率", "稼働回数", "食材原価", "労働者数", "賃金", "求人コスト", "客単価"], ["f", "i", "i", "i", "i", "f", "i", "i", "i", "i", "i", "i"], name="売上とコスト")
self.simulator.addMonitor(self.monitor6)


def step(self):

```

```

"""エージェント集合のステップ処理を行う。"""
# エージェント集合へのエージェントの追加削除を凍結
self.freeze()
# 全エージェントを凍結
if len(self.agentFreezeVars) > 0:
    for agent in self.agents:
        agent.freeze(self.agentFreezeVars)

# 内定率算出
count = 0
countframe = 0
for agent in self.agents:
    # 集計対象のみ
    (x,y)=self.env.node2xy[agent.getPosition()]
    #if x == 0 or x == 4 or y == 0 or y == 4:
    #    continue

    count += len(agent.adopt)
    countframe += agent.frame

self.adoptRate = count / countframe

self.monitor.observe(now(),self.adoptRate)

# レベルごとの採用数、応募数、枠数
adopt = {}
apply = {}
frame = {}

for agent in self.agents:
    # 集計対象のみ
    (x,y)=self.env.node2xy[agent.getPosition()]
    #if x == 0 or x == 4 or y == 0 or y == 4:
    #    continue

```

```

if agent.level not in adopt:
    adopt[agent.level] = 0
if agent.level not in apply:
    apply[agent.level] = 0
if agent.level not in frame:
    frame[agent.level] = 0

adopt[agent.level] += len(agent.adopt)
apply[agent.level] += len(agent.apply)
frame[agent.level] += agent.frame

for k in adopt:
    self.monitor3.observe(now(),k,adopt[k],apply[k],frame[k])

# 就労ログ
num = {}
for agent in self.agents:
    # 集計対象のみ
    #(x,y)=self.env.node2xy[agent.getPosition()]
    #if x == 0 or x == 4 or y == 0 or y == 4:
    #    continue

    for b in agent.adopt:
        if (b.level,b.type,agent.level) not in num:
            num[(b.level,b.type,agent.level)]=0
        else:
            num[(b.level,b.type,agent.level)]+=1

for k in num:
    self.monitor4.observe(now(),k[0],k[1],k[2],num[k])

# 内定ログ
adoptNum = 0
applyNum = 0
for agent in self.agents:
    adoptNum += len(agent.adopt)

```

```

        applyNum += len(agent.apply)

    self.monitor5.observe(now(), adoptNum,
applyNum, self.simulator.aSynchronousAgentSet.num)

    # エージェントのステップ処理
    for agent in self.agents:
        agent.step()
    # エージェントの凍結されていた変更を反映
    if len(self.agentFreezeVars) > 0:
        for agent in self.agents:
            agent.melt()
    # エージェント集合への凍結されていたエージェントの追加削除を反映
    self.melt()
def start(self):
    """同期エージェント集合の動作を開始する。"""
    # エージェント集合のプロセス
    def proc():
        try:
            while True:
                # ステップ処理
                self.step()
                yield pause(self.interval)
        except FinishException as e:
            raise e
        except Exception as e:
            fatal("[%s]エージェントエラー: エージェント集合のプロセスで
エラーが発生しました。" % self.name,
                "error in agentset process", tb = True)
    # エージェント集合のプロセスを起動
    activate(proc())
def view(self):
    """エージェントの描画を開始する。"""
    #interval = 1 # 表示間隔
    #screen = self.getAgentScreen(interval = interval, xlim = None, ylim
= None)

```

```

        #screen.addAgentSet(self)
        #screen.start()
        pass
    def custom(self):
        pass

    self.aSynchronousAgentSet2 = SynchronousAgentSet2(u"企業エージェント",
self.eLatticeGraph, simulator = self)
    self.addAgentSet(self.aSynchronousAgentSet2)

    def startSynchronousAgentSet2():
        self.aSynchronousAgentSet2.start()
        self.aSynchronousAgentSet2.view()

    self.procs.append(startSynchronousAgentSet2)

# 労働者エージェント の定義

class SynchronousAgentSet (SynchronousAgentSetBase):
    def _defAgent(self):
        class _Agent (SynchronousAgentBase):
            def initAfter(self, *args, **keys):

                # 位置を決定
                #x = random.randint(1,3)
                #y = random.randint(1,3)
                x = random.randint(0,4)
                y = random.randint(0,4)
                p = self.agentset.env.xy2node[(x,y)]
                self.setPosition(p)

                # タイプ
                self.type = next(self.agentset.typeGen)

                # レベル
                if self.type == "freeter":

```

```

        self.level = random.randint(2,6)
elif self.type == "student":
    self.level = random.randint(3,5)
elif self.type == "housewife":
    self.level = random.randint(2,5)
elif self.type == "foreigner":
    self.level = random.randint(1,3)

# 離職日数
self.elapsedNum = 0

# 就職日数
self.workNum = 0

# 採用通知まで
self.wait = 0

# 採用されている企業(企業エージェント)
self.company = None

# 状態(1:就職中、0:結果待ち、-1:求職中、-2:未就職)
self.state = "未就職"

# 色
self.screenColor = 'b'

# サイズ
self.screenSize = 30

def step(self):
    """エージェントのステップ処理"""
    if self.state == "就職中" and (self.workNum > 0 and
self.workNum % 30 == 0):
        if self.agentset.turnover[(self.type,self.workNum)] >=
random.uniform(0,1):
            self.state = "情報収集中"

```

```

        self.workNum = 0
        self.company.adopt.remove(self)
        self.company = None
        self.wait = 0

    if self.state == "情報収集中" and self.elapsedNum == 1:
        self.state = "求職中"

    if self.state != "就職中":
        self.elapsedNum += 1
    else:
        self.workNum += 1

self.agentset.monitor.observe(now(),self.agentid,self.type,self.level,self.elapsedNum,s
elf.workNum,self.state,self.wait)

```

```

    def custom(self):
        pass
    return _Agent
def initAfter(self, **keys):
    self.interval = 1.0 # ステップ間隔
    self.freezeAddAgent = True # エージェントの追加を次ステップまで凍結する
    self.freezeDelAgent = True # エージェントの削除を次ステップまで凍結する

    """エージェント集合の作成後にこのメソッドが呼ばれる。

    エージェント集合固有の初期化を行う。"""
    self.agentFreezeVars = [] # エージェントの凍結する変数リスト

    # タイプ
    self.typeGen = empiricalDistribution(weights = [[30, "freeter"],
                                                    [37, "student"],

```

```
[24, "housewife"],  
[9, "foreigner"]])
```

```
# 採否通知までの経過日数
```

```
self.notificationGen = empiricalDistribution(weights = [[5, 1],  
[10, 2],  
[15, 3],  
[40, 4],  
[15, 5],  
[10, 6],  
[5, 7]])
```

```
# 離職率
```

```
self.turnover = {}  
self.turnover[("freeter",30)] = 0.05  
self.turnover[("freeter",60)] = 0.04  
self.turnover[("freeter",90)] = 0.04  
self.turnover[("freeter",120)] = 0.03  
self.turnover[("freeter",150)] = 0.02  
self.turnover[("freeter",180)] = 0.02  
self.turnover[("freeter",210)] = 0.03  
self.turnover[("freeter",240)] = 0.03  
self.turnover[("freeter",270)] = 0.03  
self.turnover[("freeter",300)] = 0.03  
self.turnover[("freeter",330)] = 0.03  
self.turnover[("freeter",360)] = 0.03
```

```
self.turnover[("student",30)] = 0.13  
self.turnover[("student",60)] = 0.06  
self.turnover[("student",90)] = 0.05  
self.turnover[("student",120)] = 0.05  
self.turnover[("student",150)] = 0.05  
self.turnover[("student",180)] = 0.05  
self.turnover[("student",210)] = 0.05
```

self.turnover[("student",240)] = 0.04
self.turnover[("student",270)] = 0.04
self.turnover[("student",300)] = 0.04
self.turnover[("student",330)] = 0.04
self.turnover[("student",360)] = 0.04

self.turnover[("housewife",30)] = 0.03
self.turnover[("housewife",60)] = 0.02
self.turnover[("housewife",90)] = 0.02
self.turnover[("housewife",120)] = 0.02
self.turnover[("housewife",150)] = 0.02
self.turnover[("housewife",180)] = 0.02
self.turnover[("housewife",210)] = 0.02
self.turnover[("housewife",240)] = 0.02
self.turnover[("housewife",270)] = 0.02
self.turnover[("housewife",300)] = 0.02
self.turnover[("housewife",330)] = 0.02
self.turnover[("housewife",360)] = 0.02

self.turnover[("foreigner",30)] = 0.10
self.turnover[("foreigner",60)] = 0.05
self.turnover[("foreigner",90)] = 0.05
self.turnover[("foreigner",120)] = 0.04
self.turnover[("foreigner",150)] = 0.03
self.turnover[("foreigner",180)] = 0.02
self.turnover[("foreigner",210)] = 0.02
self.turnover[("foreigner",240)] = 0.02
self.turnover[("foreigner",270)] = 0.01
self.turnover[("foreigner",300)] = 0.01
self.turnover[("foreigner",330)] = 0.01
self.turnover[("foreigner",360)] = 0.01

```

# エージェントの生成数
n = 3600
#n = 100

# 1日あたりの応募人数（毎日一度も就職していない人が応募する人
# 数、離職した人は含まれない）
self.applyNum = int(n/360)
#self.applyNum = int(n/40)

# エージェントの生成
self.generateAgents(n, **keys)

#投入人数
self.num = 0

self.monitor = Monitor(["時刻","エージェント ID","タイプ","レベル","離
職日数","就職日数","状態","採用通知まで"],["I","i","o","i","i","i","o","i"],name="労働者")
self.simulator.addMonitor(self.monitor)

def step(self):
    """エージェント集合のステップ処理を行う。"""
    # エージェント集合へのエージェントの追加削除を凍結
    self.freeze()
    # 全エージェントを凍結
    if len(self.agentFreezeVars) > 0:
        for agent in self.agents:
            agent.freeze(self.agentFreezeVars)

    # 応募者の選定
    # 新規応募者を選定
    tmp = []
    for a in self.agents:
        if a.state == "未就職":
            tmp.append(a)

```

```

self.num += self.applyNum

if len(tmp) > self.applyNum:
    self.applicant = random.sample(tmp,self.applyNum)
else:
    self.applicant = []

# 求職中
for a in self.agents:
    if a.state == "求職中" and a.elapsedNum > 1:
        self.applicant.append(a)
        a.elapsedNum = 0

# マッチング
for a in self.applicant:
    # 近くの企業を探す
    #vs = self.env.neighbors(a.getPosition())
    cs = self.simulator.aSynchronousAgentSet2.agents
    companyCand = []
    #for v in vs:
    #    cList = self.env.findNodeAgents(v)
    #    for c in cList:
    #        if c.type != "company":
    #            continue
    #        if len(c.apply)+len(c.adopt) == c.frame: # 採用枠が一杯
    #            continue
    #        if a.level - 1 <= c.level:
    #            companyCand.append(c)
    for c in cs:
        if len(c.apply)+len(c.adopt) == c.frame: # 採用枠が一杯
            continue
        if a.level - 1 <= c.level:
            companyCand.append(c)

    if len(companyCand) > 0:

```

杯

```

        c = random.choice(companyCand)
        c.apply.append(a)
        a.wait = next(self.notificationGen)
        a.company = c
        a.state = "結果待ち"

# 全エージェントのステップ処理
for agent in self.agents:
    agent.step()
# 全エージェントの凍結されていた変更を反映
if len(self.agentFreezeVars) > 0:
    for agent in self.agents:
        agent.melt()
# エージェント集合への凍結されていたエージェントの追加削除を反映
self.melt()
def start(self):
    """同期エージェント集合の動作を開始する。"""
    # エージェント集合のプロセス
    def proc():
        try:
            while True:
                # ステップ処理
                self.step()
                yield pause(self.interval)
        except FinishException as e:
            raise e
        except Exception as e:
            fatal("[%s]エージェントエラー: エージェント集合のプロセスで
エラーが発生しました。" % self.name,
                "error in agentset process", tb = True)
    # エージェント集合のプロセスを起動
    activate(proc())
def view(self):

```

```

        """エージェントの描画を開始する。"""
        #interval = 1 # 表示間隔
        #screen = self.getAgentScreen(interval = interval, xlim = None, ylim
= None)

        #screen.addAgentSet(self)
        #screen.addAgentSet(self.simulator.aSynchronousAgentSet2)
        #
        #screen.start()
        pass

    def custom(self):
        pass

    self.aSynchronousAgentSet = SynchronousAgentSet(u"労働者エージェント",
self.eLatticeGraph, simulator = self)
    self.addAgentSet(self.aSynchronousAgentSet)

    def startSynchronousAgentSet():
        self.aSynchronousAgentSet.start()
        self.aSynchronousAgentSet.view()

    self.procs.append(startSynchronousAgentSet)
def run(self, **args):
    # オブジェクトの起動
    try:
        _sim_before(self)
    except:
        pass
    pass
    def startProcs():
        for proc in self.procs:
            proc()
        yield alwaysTrue()
    activate(startProcs)()
    # シミュレーション開始
    start(until = self.until, odesolver = self.odesolver, **args)

```

```

        # 結果の保存
        self.save()
        self.flush()
    try:
        _sim_after(self)
    except:
        pass
def objective(self):
    m = Monitor(name = u"充足率", basedir = self.outputDir)[u"充足率"]
    l = m.toList()
    return l[len(l)-1]

if __name__ == "__main__":
    # 実行時の処理
    simulator = Simulator(param = SimulationParam(),
        inputDir = u"input\\default",
        outputDir = u"output\\default",
        start = lambda self: None,
        warm = lambda self: 0.0,
        until = lambda self: 360.0,
        odesolver = lambda self: dopri54Solver(dt = 0.1, rtol = 0.001, atol = 1e-06))

    simulator.run()
    result = simulator.calcObjective()
    print(result)
    simulator.saveResult(result, u"objective")
    if simulator.agentApp is not None:
        print(u"エージェント画面の終了待ち受け...")
    simulator.join()
    sys.exit()

```