学位論文
ヒューマン・コミュニケーション・ダイナミクス理論

２０２２年1月

鳥取大学大学院　工学研究科
機械宇宙工学専攻　博士後期課程

松本　英博

目次

第１章　研究の背景と目的 3
１.１　背景 3
１.２　メッセージを発信するモチベーション 8
１.３　目的 8
１.４　ヒューマン・コミュニケーションとは 8
１.５　メッセージの 2 面性 9
１.６　メッセージのコンテンツの信憑性と人間関係の信頼性 9
１.７　メッセージとモチベーション 10
１.８　SNS の共通機能とメッセージング 10
１.９　広範なコミュニケーションの定義と本研究での定義 11
第２章　先行研究 12
２.１　情報理論の視点での先行研究 12
２.１.１　ネットワークとコミュニケーションの研究 12
２.１.２　社会ネットワーキングでの分析 12
２.１.３　認知科学と情報科学、さらに計算社会科学へ 13
２.１.４　先行研究での課題と本論文のアプローチ 13
２.２　モデル化の各アプローチ 14
第３章　動機（motivation）の記述とモデル 16
３.１　モチベーションのモデル化 16
３.１.１　物理情報空間とヒューマン・コミュニケーション空間の関係 16
３.１.２　会員を表すノードとリンク 17
３.１.３　イベント・ドリブン・モデルとイベント 18
３.１.４　ノードとイベント、メッセージの共有の記述 18
３.１.５　コンテンツの信憑性と人間関係の信頼性の記述 20
３.１.６　メッセージの改変と編集 21
３.１.７　メッセージングに対する戦略 22
３.２　１ノードからの情報の拡散 23
３.２.１　関数 f（.）、g（.）および h（.）の記述 23
３.２.２　初期条件 24
３.２.３　モデルの検証 25
３.２.１　M_q（t）の影響 27
３.２.２　M_q(t)、M_r(t)および M_tr(t)の変動 27
３.２.３　パーソナライズ（レベル 1） 28
３.２.４　パーソナライズ（レベル 2） 33
３.２.５　パーソナライズ（レベル 3） 35
３.２.６　パーソナライズ（レベル 4） 36
３.２.７　パーソナライズ（レベル 5） 40
第４章　個人から集団へのダイナミクス 46
４.１　同一意見の集団の形成 46

４.２　能動的ノードの利用.....47
４.３　集団のメッセージ空間.....47
４.４　集団の形成と拡散モデル.....48
４.４.１　導入するメッセージ空間.....48
４.４.２　影響度 R の範囲.....48
４.４.３　属する集団によるメッセージへの影響.....50
４.４.４　影響度 R の範囲のメッセージへの影響.....50
４.５　シミュレーションと結果.....51
４.５.１　シミュレーターの検証と集団戦略.....51
４.５.２　信頼度レベルを変える.....53
４.５.３　影響度 R を変える.....55
４.６　第３集団の動態.....61
４.６.１　同数の集団に対する第 3 集団.....62
４.６.２　同数の集団に対する第 3 のノードの R の影響.....65
４.７　寝返りとしっぺ返し.....73
４.８　インフルエンサーの導入.....74
４.９　メッセージングによる集団戦略.....75
４.１０ オピニオン・ダイナミクスとの関係.....76
４.１１ ミクロ視点からマクロ視点へ.....77
第５章　今後の展開.....78
５.１　モチベーションと意思、メッセージングの起源.....78
５.２　メッセージ集団の展開.....79
まとめ.....81
謝辞.....85
引用文献.....86
付録.....90
付録１：第３章のシミュレーションのソースコード.....90
付録２：第４章のシミュレーションのソースコード.....106
付録３：第４章のシミュレーションの動画結果.....146

第１章　研究の背景と目的

１.１　背景

ソーシャル・ネットワークキング・サービス（Social Networking Services, SNS）はインターネットを使った会員制のメッセージサービスである。

SNS は一部の ICT や情報機器を扱う専門家やエンジニアだけでなく、一般化して今や日常生活に溶け込み、インターネットでの情報交換が当たり前に行われている。図 １-1 は、国内のツイッターやフェイスブックなど主な SNS の利用率を示している。年代で見ると、2012 年の 41.4%から、2016 年には 71.2%にまで上昇していることから、SNS の利用が社会に定着してきたことがうかがわれる。

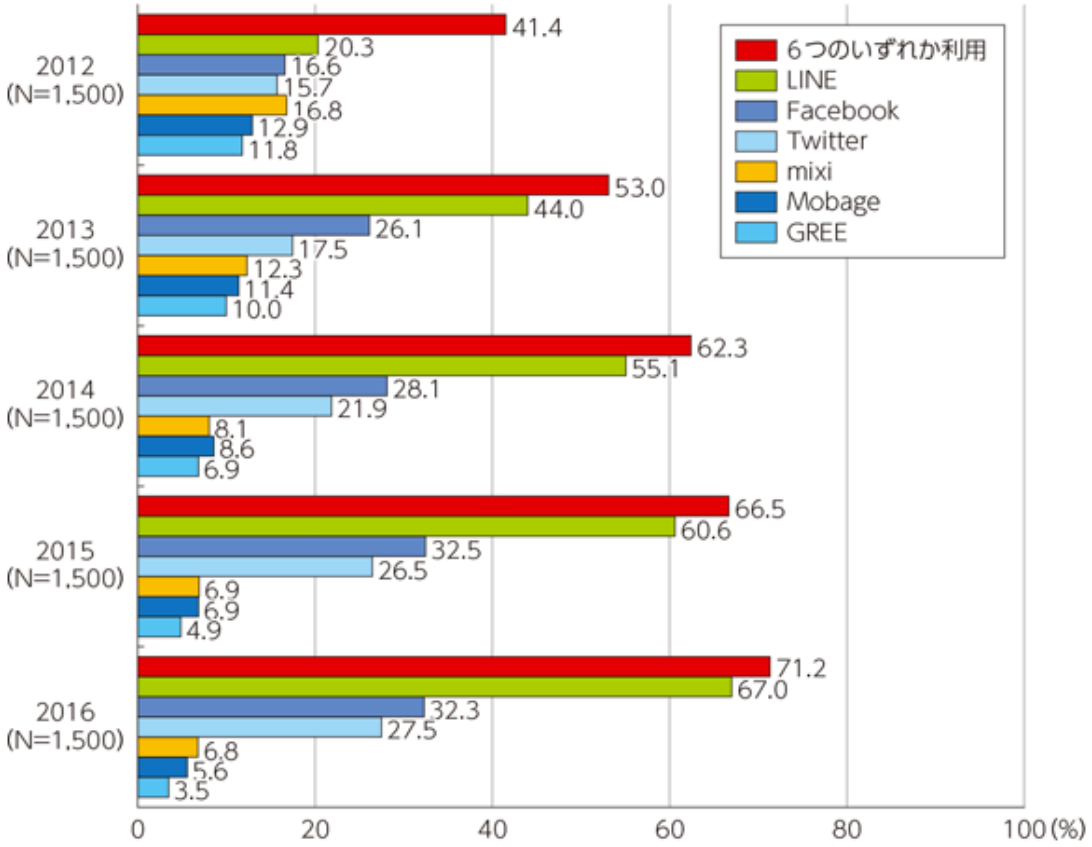


図　１-1 総務省情報通信政策研究所「情報通信メディアの利用時間と情報行動に関する調査」の「図表 1-1-1-11　代表的 SNS の利用率の推移（全体）」から抜粋[1]

さらに、図 １-2 は、世界的なモバイルインターネットの普及を示しているが、SNS はスマートフォンの普及で個人でのコミュニケーションで広く使われている。

SNS の出現以前はインターネットでの社会は専門家集団の狭い領域であったが、図 １-1 や図 １-2 の統計情報からもわかるように、SNS の出現により誰でも利用できる時代となった。さらに、SNS での個人によるメッセージはその頻度や情報の拡散といった測定可能な量でも観測できるようになった。我々はインターネットという仮想空間で社会的なネットワークを作り、利用する現象を人類史上初めて定量的に観測できる時代を迎えた。

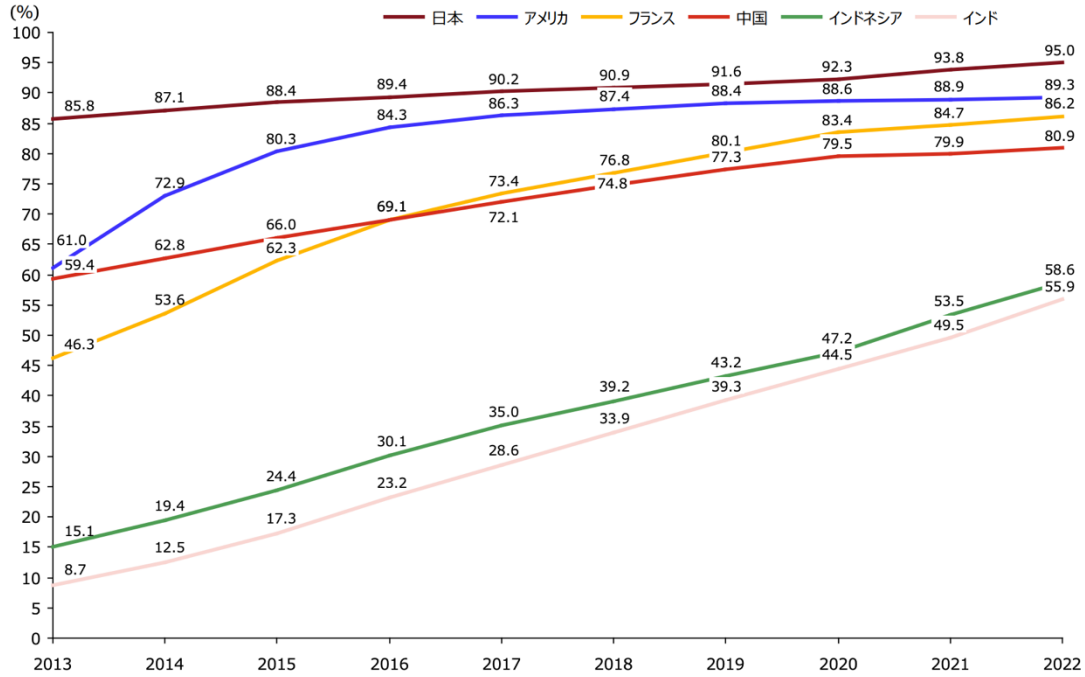


図 1-2 モバイルインターネット普及率（出典：PwC「グローバルエンタテイメント＆メディアアウトルック 2018-2022」[2]

SNS は無料で会員として登録して利用できる。登録を希望すると、メッセージを送受信するアカウントと呼ばれる会員の ID（会員番号に当たる）をサービス提供者から受け取る。ID は 1 アカウントに対して唯一で、通常は一人の個人がこれを利用する。

SNS のメッセージの送信は、ID が添付されてインターネット上のウェブあるいは専用のアプリケーションで行う。ウェブも専用アプリケーションも同じ ID でメッセージを閲覧できる仕組みとなっており、PC やスマートフォンなどの端末が利用できる。SNS はこのようにインターネットを介して各種端末で閲覧し、送受信することで、いつでもどこでも誰とでもコミュニケーションができる。

SNS は、それ以前のインターネット上のコミュニケーションサービスと何が異なっているのであろうか。[3]

相違点をまとめると、以下のようになる：

① 不特定多数の会員にメッセージを開示できる：電子メールとは異なり、指定された会員以外にもメッセージを公開することができる。SNS が世界で初めてサービスを始めた米国では、メッセージの公開を投稿（post）と呼んでいる。投稿は、例えれば、町内の掲示板の告知効果に似ている。町内の住民が掲示板にメッセージを貼り付け、掲示板のそばを通った住民や不特定の人たちに告知する。インターネット上に掲示板があり、そこに次々とメッセージを書き込むイメージである。ただし、掲示板では、匿名の投稿ができるが、SNS では匿名の投稿はできず、会員が特定される点が異なる。

② 公開されたメッセージにコメントできる：電子メールでは宛先の受信者のみが返事や回答を送信者に送り返すことができる。SNS では、メッセージを見た会員であれば特定の受信者でなくても、そのメッセージに対してコメントができる。コメントは、電子メールでいう返答や回答だけでなく、メッセージに対する感想や同意、反対などの意見も行える。１つのメッセージに対するコメント

は１つとは限らず、複数のコメントが付く場合もあり、コメントが付かない場合もある。掲示板の例えで言えば、町内の住民が掲示板の貼紙（メッセージ）を見ながら、井戸端会議をしていることに似ている。井戸端会議ならそこで話したことはその場で忘れてしまうが、SNS の場合は話したコメントが次々と掲示板に貼り付けられていくようなイメージである。

③ コメントにコメントをつけることもできる：１つのメッセージにコメントがつくと、そのコメントにさらにコメントもできる。電子メールでは、特定の相手との遣り取りが行われるが、SNS のメッセージでは、不特定多数との遣り取りが行える。

①の特徴は、電子メールなど秘匿情報に対して SNS はメッセージの公共性を前提にしている。つまり、投稿は特定の受信者を指定せず、会員のメッセージを受け取ることができる。②の特徴は、公開されたメッセージに対する社会（いわゆる世間）の反応を窺い知ることができることを示している。コメントをする受信者は、送信者あるいは投稿者からも①の公共性で知ることができるが、自分の投稿がどれほど多くの会員に影響を与えたかというメッセージ（投稿）の反応度をコメントの数やサイン（米 Facebook 社のサービスの「いいね」や米 Twitter 社の retweet の行為など、コメントでない非言語的な合図）で知ることもできる。この反応は、本研究の主題であるメッセージを送信する動機（モチベーション）にも影響を与える。多くの人に共感してもらったメッセージは発信者の大きな動機になる可能性があるからである。

１つのメッセージが反応を起こす範囲（情報ネットワーク理論[4]でいう「リンク」）が広がり、また逆に収束するのかといった条件が特定できていないという問題がある。

③は、掲示板の例のように１つのメッセージから生成される集団現象の１つである。集団現象とは、メッセージに対する感想や意見、思考や嗜好などで、受信者がまとまった行動を起こす現象である。SNS ではコメントやメッセージの発信によって、会員がその影響を受け、投稿者やコメントを行った人物像を思い描き、そのメッセージの要因や理由を求めて、コメントのコメントをすることになる。

例えば、「〇〇商店の品物 X はよかった」と A さんが投稿したら、これを見た B さんは、「確かに、〇〇商店の商品 Y もよかった」とコメントしたとする。

- A さんは B さんを同調した人物として考えるかもしれない。これに対して C さんは「〇〇商店を良く利用しているが、品物 X は△△コンビニの方が安かったよ」とコメントした。
- A さんは、高いものを買ったという後悔がおこるかもしれない。
- A さんからみて C さんは良き情報源と思うか、△△のファンで反対者とみるかもしれない。
- B さんからは A さんは、良き情報源と考えるかもしれない。
- C さんは、A さんを情報源とみる一方で、B さんは A さんの同調者とみるかもしれない。

このように A、B、C の 3 人であっても複雑な感情や動機をもつことが起こる。SNS ではこれまでの電子メールのような通信ではなく、人間によるメッセージに対する発信の動機によって、インターネットを使った社会的なネットワーク（人間関係）が形成されることがわかる。つまり、③は、メッセージの拡散を促す結果を生む。メッセージを投稿するのは１アカウントに 1 人であるから、メッセージによる衝撃

や話題性でコメントする人数も、1 人以上であれば、2 倍以上に拡散する。さらに、拡散されたコメントやメッセージを見てリンクした受信者がコメントをする。こうして幾何級数的にメッセージが拡散されることになる。[4]

SNS による仮想空間での社会にも実社会と同様に多くの課題がある。その大きな課題の中に SNS による個人に対する誹謗中傷がある。SNS の①の公共性は、受信者の不特定性と発信者や投稿者の秘匿性にあると言われている。[5]

不特定性とは、相手を指定しないで情報を公開する性質であるため、情報の拡散を意図して行われることが多い。インターネットによる広告配信でこれに関する意見を SNS で公開するといったことも情報の拡散である。

秘匿性は、受信者に対して発信者や投稿者がニックネームなど本名ではない呼称や名称を利用することで、発信者が何者であるか隠す性質である。秘匿性によって自由な情報交換ができるが、情報の内容が正しく、信憑性が高いとは限らない。

不特定な受信者に、個人情報を秘匿する発信者同士でメッセージを拡散したり、メッセージにコメントしたりする SNS では、メッセージの信憑性や受信者の信頼性に関わらず、日々情報が送受信されて利用されている。SNS の不特定多数の参画と会員の秘匿性の扱いを誤ると犯罪の道具となる一面もある。特定の会員を誹謗中傷でよって攻撃して死亡させる事件も発生している。このような痛ましい事件が起こるたびに SNS のあり方について論議がなされている。また、偽のメッセージを拡散することも SNS の不特定多数の参画と会員の秘匿性でおこる。悪意のある会員の中には秘匿性で名前を隠して偽のメッセージを発信し、不特定多数のアカウントを使って偽のメッセージの発信源が特定されないような仕掛けをした巧妙なネット犯罪も起こっている。[5]

このようなネット犯罪は、偽のメッセージを投稿（発信）したり、偽のコメントをしたり、悪意のあるコメントのコメントをすることが、③の性質で急激に広がること意図している。誹謗中傷の事件では、特定の投稿者やコメントした会員に誹謗中傷のコメントやメッセージが大量に送られることになる。偽のメッセージの事件では、事実に基づかない送信者の思い込みや偽情報（フェイクニュース）で発信されるメッセージが要因である。そこに不特定多数の参画と会員の秘匿性によって事実でないコメントやメッセージの拡散が起こる。このような偽のメッセージが SNS 上はもちろん実社会でもデマを生み、これに基づいた行動で時には犯罪や事件の引き金になる場合もある。受信者が個人ではなく、企業の SNS 担当者であれば、企業のイメージや商品のブランドを傷つける可能性もある。[5][10]

③の性質は負の作用だけでなく、正の作用も存在する。例えば、ビジネスにおける商品やサービスの「口コミ」が SNS で広がることで、ヒット商品が生まれる現象がある企業のマーケティング担当者は、口コミが自社の商品やサービスが他社よりも優位になるように SNS での有力なメッセージの拡散者に自社の商品やサービスを好意的な取り扱うように支援する。このような情報の拡散者はインフルエンサー（influencer）と呼ばれ、インフルエンサーの SNS での仲間がメッセージをさらに拡散する。企業にとっては有意義で正の情報の拡散となるが、現象やメカニズムはデマの拡散と「口コミ」のマーケティングによる拡散と同じである。インフルエンサーまたは犯罪者が発信するメッセージが社会経済に有害か無害かの違いとも言える。**エラー! 参照元が見つかりません。**[16][17]

SNS のもつ誹謗中傷や偽のメッセージの課題は根本的にどのように考えれば良いだろうか。

性質①について考えてみると、不特定多数ではなく、特定化することが考えられる。具体的には、該当するメッセージを閲覧できる SNS の限定会員の中でサービスを提供することで、少なくとも SNS の管理者には犯罪発生時など特定できることで対策が打てる。会員の秘匿性を担保しながら、匿名のままでメッセージに犯罪につながるようなコンテンツ（内容）やコンテクスト（文脈）を含んでいないかを検知して、SNS の管理者は問題のあるメッセージを削除する。偽のメッセージについては、メッセージを監視して SNS の外部にある情報源（メディアなど）と比較し、犯罪に絡む場合メッセージの削除通知を投稿者に送る。SNS で犯罪に絡んだコメントが拡散してしまった時は、SNS の会員が問題のコメントを利用端末に保存し、これを拡散してしまう場合もあることから、メッセージを全て削除することは難しくなる。

性質②や③を見ると、誹謗中傷の対策として問題のメッセージを拡散するようなコメントをさせないことも初期の犯罪防止対策となる。例えば、コメントを受信者が特定の ID を持つ会員のみに限定するといった対策も考えられる。不特定多数から限定され素性の分かっている会員同士であれば、誹謗中傷の抑止につながるという考え方である。また、コメントの内容で誹謗中傷の報告を受けて SNS 運営者の権限で、コメントを削除する方法もある。

しかし、最も有効と思われる方法は、コメントを行う際に犯罪に繋がる可能性もある誹謗中傷を許さないといった意思や動機をコメントやメッセージの送信者が持つことである。

偽のメッセージも事実でない情報が拡散されることであるが、性質①の最初のメッセージや性質②や③でのコメントが、信憑性が低く、真偽も事実に反していても、会員に拡がることを面白がる「悪戯」で、偽のメッセージが「口コミ」されやすい状態となる。最初のメッセージが悪意によって意図的に事実無根の「偽のメッセージ」から始まると、投稿者を信頼する会員でメッセージが事実であるかないかに関わらず、「善意」でこの情報を早く伝えねばといった危機感や使命感でコメントしてしまう。危機感や使命感が高ければ高いほど、事実確認を SNS 以外の外部情報で確認しないまま、コメントしてしまうことになる。誹謗中傷の現象と同様にコメントやメッセージの送信者が、事実でない事項を投稿しない、コメントしない、広げないといった意思や動機を持つ必要があろう。

このように、メッセージを投稿し、コメントをする意思や動機が、SNS を利便性のあるコミュニケーション手段とするか、誹謗中傷や偽のメッセージといった犯罪につながる悪の道具とするかを左右することがわかる。コメントやメッセージをどう取り扱うかを考えることは、SNS が社会に利益を生むか利便性のあるものにするか、ネット犯罪の元凶を生み出すものになるかといった社会課題の解決の一助になる。コメントやメッセージをどう取り扱うかを考えることは SNS の管理者の視点であるが、会員の視点でみると、さらに根本的な「なぜメッセージを送り合うのか」という基本的な命題を解くことに遭遇する。背景がわかれば、SNS の課題を解決するヒント、示唆や要点を得られると期待できる。

本研究では SNS で扱われる情報を特定相手に発信される情報と区別するためにメッセージと呼ぶ。また、メッセージの内容をコンテンツと定義する。

本研究の背景は、現実での出会いなしにインターネットでのみ知り合っているような相手に対して「なぜメッセージを送り合うのか」という根本的な疑問を解くこ

とである。この疑問を解くことは、SNS 上で起こる事件や課題に対する対策を見出す一助になり、有意義な SNS の応用を導くことになるからである。

1.2　メッセージを発信するモチベーション

SNS は、メッセージする投稿する行為、メッセージの内容であるコンテンツと全会員を含む情報空間で構成されている。会員はコンテンツの送受信の媒体であり、コンテンツの生成、編集者である。コンテンツは、会員が取り上げる話題、例えば、社会・経済・教育・ビジネス・生活といった話題から作り出される。コンテンツは、さらに、送信者の動機と送信者（投稿者）の影響を受ける受信者への信頼性や他の会員への影響の大きさを判断して作成、編集される。投稿者の動機（モチベーション）によって、このような個人的でミクロなコンテンツが、SNS の情報空間に拡散され、マクロなダイナミクスを生み出す。

1.3　目的

本研究の目的は、SNS のようなヒューマン・コミュニケーションの情報空間でのダイナミクス（Human communication dynamics）を明らかにすることを目的としている。この目的を達成することで、SNS の持ついくつかの課題を解決する一助とする。

1.4　ヒューマン・コミュニケーションとは

一般的なネットワークサービスは、データ通信網やインターネットのように人間の介在を必ずしも必要としない。本研究の対象は、人間の対話、伝言、口コミなど人間のモチベーションや欲求によって情報を生成、編集、送受信する行為であり、ここでは、一般的なネットワークサービスに対して人間が介在する意味でヒューマン・コミュニケーションと定義する。

情報通信技術（ICT）でのヒューマン・コミュニケーションは、発信者の主体的で個人的な意見の存在が必要である。例えば、業務上の電子メールでは、同じ人間同士の通信ではあるが、業務遂行の目的で利用されるために発信者の主体性はあるが、個人の思考はあっても思いは重視されない。また、受信した文脈（context）に沿わない情報の発信は皆無であろう。これに対して、SNS は、基本的に発信する情報は自由であり、受信したメッセージに無関係でも発信されることがある。

SNS はインターネット上のサービスであることから従来のネットワークサービスをインフラストラクチャー（情報基盤）で利用される。この情報基盤で人間同士の会話や投稿、コメント、反論、「いいね」などの感情反応などがやりとりされる。これらのやりとりは、ヒューマン・コミュニケーションの 1 つだといえる。

これまでの情報工学や情報ネットワーク理論では情報基盤についての研究対象が多く散見される。興味深いことに、この情報基盤上のコミュニケーションであるヒューマン・コミュニケーションについては、近年、人工知能（AI）研究の分野で人工自我などの研究でようやく取り上げられ始めたばかりである。[6]

日本では明治時代に information を「情報」と翻訳して以来、「情」よりも「報」に重点を置き、情報科学やコンピューター・サイエンスが発達してきた。[3]

さらにインターネットの世界的普及により、誰でもどこでも誰とでも個人が意見やコメントを全世界に発信できる環境が提供されてきた。「情報」の「報」だけでなく、「情」である人間の感情や行動、活動を世界に発信できる時代になったといえ

る。ヒューマン・コミュニケーション・ダイナミクスは、「情報」の「情」の時代に存在する人間のダイナミクス（動的な状態変化）を説明する「人の情報工学」であることを本研究で示したい。

1.5　メッセージの 2 面性

本研究の対象となる SNS では「情報」の「情」をメッセージで伝える。

例えば、「明日、〇〇商店で安売りがある」というメッセージが SNS に投稿されたとしよう。このメッセージに信憑性があるかないかに関わらず、会員は自らの判断で、「安売り歓迎！〇〇商店は肉屋だからすき焼き用の牛肉は安くなるのだろうか」といったメッセージをコメントする。このコメントは会員に向けて、以下のようなコンテンツを発信していると考えられる：

① 〇〇商店は安売りを行う
② 〇〇商店は明日安売りを行う
③ 〇〇商店は肉屋である
④ 投稿者はすき焼きの牛肉に興味がある
⑤ 投稿者のメッセージで〇〇商店の安売りの対象にすき焼き用牛肉が含まれているか疑問がある

といった情報を含んでいる。いくつかのコンテンツは事実かもしれないが、そうではないものもあり、さらに⑤のように投稿されたメッセージだけでは判断がつかないものも含んでいる。確かに、情報理論で考えると、①から⑤はメッセージを発信している事実は記述できるが、コンテンツに関しては、曖昧である。認知科学やマーケティングでは、①や②、⑤は、安売りの肉への購買欲求が高まるが、③や④は受信者の購買条件でしかない。このように、メッセージの 1 つの面は、コンテンツを運ぶという役割である。またもう 1 つの面は、コンテンツによって受信者の行動やモチベーション、感情を左右する役割である。

ヒューマン・コミュニケーションの動態を考える上で、メッセージの 2 面性を理解して、情報の「報」だけでなく、「情」も対象にする必要がある。

情報の「報」に関しては情報理論の分野で多くの先行研究があり、情報の「報」に関しても情報発信についてモチベーションなどの視点から社会学や認知科学の分野で先行研究がある。しかし、この 2 つの架橋となる分野に関しては未開拓領域があり、本研究で取り扱っていきたい。

1.6　メッセージのコンテンツの信憑性と人間関係の信頼性

コンテンツの信憑性（reliability）とは、メッセージの内容が事実に反するかどうかの指標と定義する。信憑性は、情報源に依らずメッセージの内容で判断する。

一方、受信者の信頼性(credibility)とは、メッセージの発信者と受信者の間の信頼関係の度合いで、信頼性が高いとは、発信者側から見て受信者を信頼していると定義する。

信頼性は、受信者側から見た信頼度ではなく、メッセージが正しく伝えられる受信者らの割合である。SNS では不特定多数の受信者であることから、信頼性が高い状態とは、発信者と受信者が相互に信頼している場合が最大となる。相互の信頼も発信者と受信者が現実の生き方（real life）で出会うことがなくてもインターネットのコミュニケーションの頻度や深さから相互の信頼が生まれていく。[9]

1.7　メッセージとモチベーション

SNS で発信されるメッセージは自発的で発信者が受信者に何らかの行動を促したり、思いを伝えたり、他者からの意思表示を期待している。受信者は発信者のメッセージによって意図や意思を感じ、例えば、メッセージに同調したり、反対したり、黙認、拒否さらに無視といった判断を行う。SNS では、会員での「友達関係」で知り合いであれば、受信者のメッセージも確認でき、発信者は自分の発信したメッセージの影響を知ることができる。これが発信者にとっては、次のメッセージを発信するモチベーションにつながる場合もある。

発信者は、SNS 以外からの情報でメッセージを発信する場合もある。例えば、前述の肉屋の安売りの例であれば、折り込みチラシの広告などで知った情報からメッセージを発信することも SNS 以外からの情報になる。このような SNS 以外の情報源を本研究ではイベント（event）と定義し、SNS 内のメッセージと区別する。この区別によって、SNS の外部情報と内部での交流による効果を分けることができる（参照　3.2）。デジタル・マーケティング論では、人為的に宣伝広告する商品に関するニュースやキャンペーンをマスメディアでイベントを設定し、SNS のメッセージの拡散で「口コミ」を広げ、広告宣伝の効果をあげる方法も利用されている。[10]

1.8　SNS の共通機能とメッセージング

今や多くの SNS が存在する中、本研究で扱う SNS はどのようなものかを示す。

SNS の起源は、多くの解説によれば日本国内での mixi や米国の Twitter などにあるといわれている。多くの SNS に共通して機能を列記すると以下のようになる：[7][7]

- プロフィール機能：会員の個人情報（プロフィール）を会員の意図で共有する機能。通常は他の会員には秘匿性が守られる。
- メッセージの送受信機能：会員同士のメッセージ交換を行う機能
- タイムライン（ウォール）機能：メッセージの投稿やメッセージに対するコメントを時間経緯に従って共有する機能
- 会員招待機能：会員の招待によって新規に会員を SNS に登録する機能
- 会員相互リンク機能：会員同士が双方向のリンクする機能
- 会員検索機能：SNS 内の会員を検索する機能
- ブログ機能：メッセージを保管して他の会員と共有する機能
- 質疑応答機能：特定のメッセージをテーマに質問したり、回答したりする機能で送受信機能の特殊な利用法
- アンケート調査の機能：メッセージで会員にリンクした会員を相手にオンラインでアンケートをとる機能
- コミュニティーを形成、管理する機能：同じ趣味や同好などグループを形成し、管理する機能

にまとめられる。[7]

本研究の目的は、SNS の歴史を追うものではないので、一般的に普及した SNS で種類や主催者によらず、最も基本的であるイベントやメッセージ、さらにコメントが会員のタイムライン（ウォール）機能を単純化したメッセージシステム（メッセージングモデル）を扱う。（前述の多くの機能については引用文献を参考にされたい。[7]）

1.9　広範なコミュニケーションの定義と本研究での定義

情報理論や社会学、認知科学のみならず、コミュニケーションの定義は人類の歴史とともに変遷した。人と人が対面し、相手の肉声を直接聞いたり相手の顔を直接見たりして行うコミュニケーションが基本で、人類の歴史の初期から行われている。古代から手紙のように遠隔地にいる人とコミュニケーションをとることも広く行われている。また 19 世紀ごろからは電信や電話など、電気的な技術を用いて遠隔地の人とコミュニケーションを行うこともできるようになった。直接対面せず、離れた場所にいる人とコミュニケーションを行うことを通信（tele-communication）としている。[18]

コミュニケーションに使われるメッセージも言語を利用するものや非言語を利用するものも取り扱われている。SNS では「絵文字（pictogram）」と言われるセマンティックス（意味論）をもつ言語と非言語と中間な形態も存在している[18][17]。

コミュニケーションの機能は、単に情報の伝達にとどまらず、情動的な共感、さらには相手の行動の制御をも幅広く含んでいる[18]さらにコミュニケーションが成り立つ条件は、そのための適切な発信が行われたというだけではなく、受信者が適切なシグナル、メディア（媒体）、イベントに注意を向け、情報を受信した上で、さらに的確な理解をしているかどうか、という点にもかかっている。記号の解釈にあたっては、相補的関係にあるコンテクスト（非言語的な文脈）とコード（言語的な約束）とが参照される。[18]　定められたコードを参照するだけでは、メッセージが解読できないとき（たとえば子供のコミュニケーション）、コンテクストが参照され、受信者による推定が加わる事になる。[18]

本研究では、人間同士の交流に必要なモチベーションを考察するために、広範なコミュニケーションの定義を扱わず、狭義で根源的なコミュニケーションの機能に限定することで、本研究の目的であるヒューマン・コミュニケーションの動態を扱うことにした。コミュニケーションを発信と応答という観点から観察し、ある個人の発信に応じて別の個人が応答する場合、両者の間にコミュニケーションが成立しているとするという定義である。

第２章　先行研究

ヒューマン・コミュニケーションに関連するこれまでの研究について要点を述べる。先行する研究の問題点と課題を示し、本研究が焦点を当てる命題を示す。

本研究で注目するヒューマン・コミュニケーションでは、コミュニケーションの手段や方法に注目する情報理論からの視点とコミュニケーションの主体である人間の動機（motivation）や意思（intention）からの視点の両者の立場で検討する必要がある。

一方、両者の立場を考慮したモチベーションによる情報理論を可視化して検討できるモデルの必要性が出てきた。

２.１　情報理論の視点での先行研究

２.１.１　ネットワークとコミュニケーションの研究

大向のコミュニケーションツールから情報の基盤への視点からの解説では、SNS をコミュニケーション分析、社会ネットワーク分析、及び情報・知識共有の 3 つの観点に基づいて分類している。[7]　この中でコミュニケーション分析では、心理学や社会学、近年は行動の心理学といった分野で、SNS におけるコミュニケーションの特性を明らかにしている。解説には mixi の参加者にアンケート調査を行って、本論文の焦点でもある利用目的や個人情報の開示について心理とその傾向を分析している。また、海外での SNS でのコミュニケーションが文化的違いで異なることも分かったという。さらに、社会ネットワーク分析の視点では、人間の間のリンクに注目し、従来は上述のアンケート調査などコストの面と大規模なネットワーク上ではデータ取得でさえ困難であったが、投稿記事の分析などからネットワーク構造を研究することが出来るようになった。この分野では SNS の出現以前もすでに Wellman らの論文を皮切りに研究されてきた。[20]

２.１.２　社会ネットワーキングでの分析

社会ネットワークでの分析では、SNS の会員を本論文と同様にノード（node）とし、中心性と呼ばれる指標を設定して計算する。例えば、ノードの持つリンク数（次数）と任意のノードとの平均距離、固有ベクトル、ネットワークの密度（クラスタ係数）などで、どれもネットワークの構造を比較分析するものである。興味深いことに大向らは、SNS はランダムネットワークではなく、知人同士が密接につながった小規模コミュニティーが、リンク数の多いノード（ハブと呼ばれる）によって大域的に連結された構成であると予想している。[4]　この予想に沿うように、Barabasi らが優先選択モデルを提案して、スケールフリー構造に着目して大規模複雑ネットワークの形成まで論じている。[7]　スケールフリーネットワークとは、ノードとリンクの関係が、規模にかかわらず同様の冪分布をすることで、小規模コミュニティーと大規模な複雑ネットワークを結びつけようという考えである。

大向の解説にある湯田らの研究は、mixi を使ったデータで、次数の低いスケールフリー性と高い凝集性が認められたことを報告している。[7][29]　同報告では局所的にリンクが強い集団を高密度集団と呼び、統計的経験則として知られている Zipf 則による予測値と比較分析している。なお、Zipf 則とは、出現頻度がk番目に大きな要素は１位の要素の頻度に対して、$1/k$に比例する経験則である。同報告の結果は、全体の集団構造ではおおよその Zipf 則に沿った分布をした。つまり、集団の高密度

集団の大きさごとに分布すると思われたのに、100 人から 300 人ほどの高密度集団が明確に現れず 100 人から 300 人以上へジャンプするという現象を発見している。その要因としてクリーク（徒党を組む）現象ではないかとしている。[29] 社会ネットワークでは、SNS の会員の意思を間接的に、リンクによるネットワーク構造に求めて、その性質を解析している。

社会ネットワークを情報理論でのネットワークの性質を拡張して理解することは、今後も重要であるが、本論文で述べている「情報」の「情」へ領域には湯田らの研究のように、理論的にクリークのような現象が説明できないところに課題がある。推測ではあるが mixi の例ではどうやら日記機能が別の会員の興味をそそることで、高密度集団の生成が予測つかないところで起こっているという。このような現象は、本研究で取り扱っている会員同士のモチベーションによって親密度が上がるが、大きな集団にはならず、影響範囲が小さい集団での振る舞いとして考えられる。社会ネットワークのアプローチはマクロ的であり、会員レベルのミクロとの架橋を行なう必要性を感じる。

2.1.3 認知科学と情報科学、さらに計算社会科学へ

一方、ヒューマン・コミュニケーションで主体である人間は、メッセージの受信者でもあり発信者でもある。さらに、人間は動機や意思を持ってメッセージの内容（コンテンツ）を記憶する。さらに記憶したコンテンツを編集する。発信者は記憶したコンテンツを自ら決めた信頼した受信者に編集し、投稿あるいはコメントする。編集の程度は、発信者の動機や意思によって異なる。受信したコンテンツを記憶に留めるだけで、受信者に送信（投稿）しない場合もある。このような心理的な決定メカニズムは、行動の心理学分野で研究されてきた。[30]

コンテンツは嗜好や性格といった個人の特徴に影響を受け、会員同士のコミュニケーションに影響を与えると考えられる。

人間のコミュニケーションの動機に関するこれまでの研究は、認知科学の分野で扱われている。[42][43] ただし、定量分析には適していなかったが、岡田らの認知心理学の研究は、曖昧模糊とした心の働きと情報理論との架橋となり、モチベーションもこの範疇に入る。[8][30]

石井、鳥海らの著書[8]によれば、認知科学が人工知能などの応用で脚光を浴びたころ、社会ネットワークの研究との交流で世界的な規模で、ノースイースタン大学教授で政治学者の David Lazer らの 2005 年の論文[21]が起源となって総合的な科学である計算社会科学が生まれたとされている。[8] 計算社会科学では人間や社会の理解のためには、社会ネットワークと社会的相互作用の理解が本質であるとされ、SNS や多くの新しいデータやツールが重要な素材と方法になるという。本論文の関連分野は計算社会科学に含まれる。[35][36]

2.1.4 先行研究での課題と本論文のアプローチ

先行する研究での多くの課題の中で本論文はどこを狙うのかを以下に示す：

- 社会ネットワークの研究では、コミュニケーションを促す存在が研究の対象ではなく、コミュニケーションが存在することを前提としている。コミュニケーションが必要だから、ネットワークがあると見るか、ネットワークがあるからコミュニケーションをするのかは不問である。本論文では、モチベーションがあるから、コミュニケーションをネットワークで行うと考える。[17]

- 情報理論では、SNS で流れる情報に種別はなく、広く情報網を使うために均一なデジタル情報基盤でのコミュニケーションを取り扱い、人間の会話であれ、メッセージであれ、センサーネットワークのデータであっても同一である。「情報」の「報」が主軸である。SNS で誹謗中傷の投稿があっても、情報理論では音楽のストリーミングで流れるデータ群でしかない。本論文では「情報」の「情」の領域に関わる。
- 計算社会科学では、まだまだ分野の広さ故にそれぞれの分野の結節点での成果が多くはない。[8] 本論文もその成果を導く一助としたい。

2.2 モデル化の各アプローチ

メッセージを投稿することをメッセージングというが、メッセージング自体を数理モデル化する試みを始める前に、人間のコミュニケーションが、観測可能なモデルに置き換えられないかを検討した。もし置換できれば、コンピュータでシミュレーションでき、SNS への応用も可能となる。メッセージングと機能面をモデル化するために表 2-1 で代表されるモデルを適応することを試みた。

情報技術の分野では、情報とエネルギーの伝達は、時変係数確率モデル[11]、伝達関数モデル[14]、エネルギー伝達モデル[14]、分散モデル[14]などで説明できる。ただし、人間の感情を含むメッセージによる自発的相互作用（メッセージング）の分析は、各モデルの抽象化する段階では扱われていない。

表 2-1 *メッセージングを表現する各モデルの特徴*

	SNS との親和性が高いと思われる特性	SNS との親和性が低いと思われる特性
時変係数確率モデル[11]	伝播の拡大、収束の要因仮説が立ちやすい	拡大、収束の要因がどのメッセージでも同じであるか特定できない
伝達関数モデル[14]	メッセージごとの伝播特性が捉えられる	メッセージの特徴を記述して分析することが困難である
拡散モデル[27]	メッセージごとの伝播特性が捉えることができる	伝播特性から逆にメッセージを特定できない
浸透モデル[4]	マクロなメッセージの伝播特性は捉えやすい	爆発的な拡大といった伝播特性が解析できない
意思決定による通信モデル[12]	決定木などで伝播プロセスは記述しやすい	意思決定の要因分析が複雑になる
エネルギーの伝播モデル[13]	メッセージのモチベーションなど伝播の規模や時間特性は記述しやすい	メッセージとエネルギー伝播の関係が不明確になる

同表での時変係数確率モデルは統計的解析のモデルとして使われ、特に多くの状態変数をシステムとみてマクロ的なフィードバックシステムに置き換えることもある。SNS でノードの状態を状態変数に置き換え、メッセージをシステムのフロー

（流れ）と見ることで、メッセージの伝播の拡大や収束を観測する。フィードバックシステムと同様に、負帰還を状態変数にかけることにより制御できるか否かをみる。確かに可制御状態であれば、伝播の拡大や収束に必要な負帰還をかければ良いとわかるが、実際は膨大な変数をテンソル解析するために時間とコストがかかり、しかもフローに依存するために一般解が得にくいという欠点がある。

伝達関数モデル[14]は、メッセージを時系列の入力ベクトルと捉え、出力をメッセージの拡散と見た場合のマクロ的な関数を得て、未知の入力に対しても出力を推論できるモデルである。特徴は表 2-1 にあるようにメッセージごとの伝搬特性を得られるが、その特性が何を示すのかを特定するのは抽象的で扱いにくい。

拡散モデルと浸透モデルは、メッセージの拡散と物理的な拡散現象の相似性を的確に使ったモデルで、現象を記述するには浸透モデルも同じく簡便であり、マクロ的な拡散あるいは浸透を観測するには適しているが、拡散モデルでは拡散現象の要因については特定しにくい。

ここで、拡散モデルは、本論文の３．１．５の数理モデルとしても応用することから紹介しておく。(a)式は石井らが拡散モデルを基にして、ノードi,jとkの 3 者におけるツイッターのつぶやき（メッセージ）を示した数理モデルである。[27]

$$\frac{dI_i(t)}{dt} = C_i A(t) + \sum_j B_{ij} I_j(t) + \sum_j B_{ij} I_j(t) \sum_k D_{jk} I_{jk}(t) \qquad (a)$$

まず、(a)式の形成過程でこれを説明する。

時刻tでメッセージが投稿された数をI_i(t)とする。ただし、投稿数全てではなく、あるイベントやメッセージに対するノードiへの投稿数である。イベントに対する外部イベント情報の大きさ$A(t)$とすると、会員によって影響される割合C_iを考慮して

$$\frac{dI_i(t)}{dt} = C_i A(t)$$

と書ける。ノードjに対するコメントをする確率をB_{ij}とすると、

$$\frac{dI_i(t)}{dt} = C_i A(t) + \sum_j B_{ij} I_j(t)$$

となる。第２項は、jに関わるコメントの総数である。(a)式の第３項以降はコメントのコメント数、さらにコメントのコメントに対するコメント数というように再帰的に起こった場合の項である。ここでの課題は、（a）式によってメッセージの拡散は定量的で計測可能な現象、例えば、映画などのヒット現象を SNS でマクロ的に捉えモデルにするには適しているが、拡散の要因であるメッセージの投稿I_i(t)やノードjに対するコメントをする確率B_{ij}が、ノードiの何から発生したのかは語ってはいない。

その他のモデルとして、メッセージをロジックツリーで拡散を説明することや原子物理のエネルギーホッピングにヒントを得て、ノードを一種の量子井戸と見るアプローチも存在する。[13]

先行する多くのモデルで SNS におけるモチベーションによるメッセージングで適応可能かを調査し、モデル化の試行を行った。しかし、どのモデルであっても前提条件として、SNS を使うことが人間の行動である限り、人間（心理）自身をモデルの対象として組み込む記述が必要であることがわかった。具体的なモデル化は、第３章で示す。

第 3 章　動機（motivation）の記述とモデル

本研究では、ヒューマン・コミュニケーションを情報科学と認知心理学を架橋した視点で捉えることで、ヒューマン・コミュニケーションのダイナミクスを論じる。準備として、ヒューマン・コミュニケーションのモデル化を考え、目的であるモチベーションの影響を考察する。

3.1　モチベーションのモデル化

SNS の会員とそのメッセージをどう情報理論と行動心理のアプローチでモデルに反映させるのか。これが本研究での第 1 の大きなハードルである。第 2 のハードルは、そのモデルの検証である。

第 1 のハードルを越えるためには、ヒューマン・コミュニケーションを情報の「情」と「報」で記述できる必要がある。「報」の領域を情報理論のモデルで記述し、「情」である会員の心理を情報理論のモデルのパラメータや変数に適応させる方法で SNS の会員とそのメッセージの関係の両方を記述することで解決しようとしている。第 2 のハードルを越えるには、そのモデルが、情報理論に沿い数理モデルで記述でき観測できねばならない。人間行動を観測できるパラメータに置換することでモデルの検証が期待できる。以下、このアプローチで進めることにした。

3.1.1　物理情報空間とヒューマン・コミュニケーション空間の関係

SNS は、会員とその知人、その間で送受されるメッセージを SNS 内で共有する掲示板のような情報空間から構成されている。ここでいう情報空間とは、情報の送受信、伝搬に人間が関与せずに動作する「物理情報空間」とは別の人間同士でなければ意味をなさない「ヒューマン・コミュニケーション空間（人的情報空間）」を指している。

ヒューマン・コミュニケーション空間は、電子メールと手紙で比較して考えるとわかりやすい：

① 電子メールも手紙も送る内容は人間同志にしかわからない
② 電子メールも手紙も送る相手は人間であればその反応が予想でき、返答が期待できる
③ 電子メールも手紙も複数の相手に複写などを行なって送ることができる

と役割として差異はない。しかし、電子メールと手紙は、これらを扱うインフラストラクチャー（情報基盤）に大きな違いがある。電子メールは情報理論に則った電子機器や回線、電気信号といった物理的情報空間をインフラストラクチャーにしているが、手紙は郵便制度という集配配送によるインフラストラクチャーを使っている。前者は速度やコスト、利便性を考えると後者より優れている。しかし、人間は非効率だとして前者のみ使い、後者を捨ててはいない。この不自然さが人間のコミュニケーションの特徴で、家族や友人、恋人など相手に応じて、封書の内容を変えたり、タイプではなく手書きで手紙を書いたりする行為で、物理的情報空間のみでは説明できない付加的な情報が存在する。これが人間のもつモチベーション（動機）で促進される。[26]

仮説として、このような情報空間を「ヒューマン・コミュニケーション空間（人的情報空間）」と本論文では定義している。

SNS は、インターネットという物理情報空間に「ヒューマン・コミュニケーション空間（人的情報空間）」が重畳された情報サービスである。インターネットがインフラストラクチャーとして一般に普及する以前は、人的情報空間は、井戸端会議や口コミといった「噂」から電話やマスメディアと言われる、テレビやラジオ番組、広告宣伝や新聞、書籍や雑誌まで、データを扱う物理情報空間とは別に存在していた。詳細はマスコミュニケーション論で説かれているが、放送と通信の融合が始まって以来、インターネットの普及で融合から、統合に発展し、物理情報空間とヒューマン・コミュニケーション空間は一体化した。第２章先行研究にあるように、日本では mixi、欧米では Twitter や Facebook が物理情報空間とヒューマン・コミュニケーション空間は一体化した情報空間を提供し勃興した。[3]

以下の議論では特に断らない限り情報空間を物理情報空間とヒューマン・コミュニケーション空間が一体化したものとする。

3.1.2　会員を表すノードとリンク

SNS の会員はそれぞれ独立したモチベーションを持ち、会員のメッセージは SNS を使う時の情報空間の状態に依存している。まず、この会員を情報空間で表す必要がある。情報ネットワークの中継点やサーバーといった情報理論で扱うノード（node）で、SNS の会員を表わせないか試みてみる。つまり、情報ネットワークを構成するノードを SNS の会員とみなし、ノード間のリンクを使ったネットワークを利用すれば、複雑ネットワークの研究手法と同様に定量的に動態を考察できる可能性がある。

情報ネットワークでノードとリンクを置き換えて、SNS を考えるには、幾つかの簡素化と読み替えが必要となる。

簡素化とは、実際の SNS の設定にはあるが、本研究のヒューマン・コミュニケーションに関わらない部分を省略することである。例えば、実際の SNS では、会員 ID を取得したときに SNS のサービス提供者との連絡などの設定があるが、これらの設定は、ヒューマン・コミュニケーションの動態とは無関係であるので省略する。同様な簡素化には、会員の入会制限の撤廃である。これは、SNS が黎明期の頃に採用されたルールで、新規会員になるためには既存会員からの招待がないと会員 ID が設定できないルールである。今回は入会制限を設けず、新規会員は、会員の誰とも交流がない状態から始められると考える。[17]

読み替えとは、複雑ネットワークでのノードを SNS の会員、リンクを会員同志の繋がりとして理論を展開することである。

次に他の会員との交流はどのように表現すればよいだろうか。

新規会員でも、SNS で表示され共有されるタイムライン（時系列の表現）などのように他の会員の公開メッセージを共有し閲覧できる機能が使える。例えば、新規会員 x さんは、リンクがなくても公開メッセージは、メッセージの閲覧機能を使って閲覧できる。自分以外の会員間での公開メッセージを閲覧しながら、自分の嗜好によってメッセージを投稿している会員 y さんを選び、仲間になるために x さんの投稿のコメントを書き込むか、非言語系の「いいね」のような反応する。この状態で会員 x と会員 y は接触したことになり、繋がり（リンク）が生成されたとみなす。このリンクの定義では電子メールのように決まった特定の受信者に情報を送るような確実で強いリンクではなく、会員 x と会員 y はメッセージを中心にした一時的で不安定な弱いリンクができる。メッセージの電子メールと SNS との扱いの違いは受

信者が特定できるか否かにある。電子メールであれば、受信者は確定しており、メッセージも相手に応じてコンテンツを変えることができる。SNS の場合は、不特定で多数な受信者の好みや状況など特性を考えたコンテンツでなければ、メッセージは無視される場合もある。

3.1.3　イベント・ドリブン・モデルとイベント

実際の SNS に近く、モチベーションによるメッセージングが記述できるモデルを選択する必要がある。実際の SNS でのメッセージの送受信を見ると、イベント・ドリブン（駆動）で説明できる。

例えば、会員 A さんがサッカーチーム X のファンで、テレビでチーム X の試合を見ながら SNS を使っているとしよう。X の選手がゴールを決めて喜びを伝えたいと「チーム X の素晴らしいゴールが決まった。」とメッセージを投稿したとする。さて、この場合、サッカーの試合の様子は SNS の外部の情報であるが、A さんのモチベーションによって、結果の感想を投稿しようとメッセージを考え、即座に投稿した。投稿によって、試合を見ていない他の会員にも広がっていく。

外部からのメッセージを送信（投稿）する契機は、サッカーの試合の様子だけでなく、ニュースやゴシップなど多くの情報源がメッセージの契機となる。このような情報源の契機を（SNS 外部の）イベントと呼ぶことにする。

これに対して、A さんのメッセージを見た会員 B さんは、A さんのメッセージが引き金となり、「やった！今日の試合は X の勝ち！」とメッセージを出せば、A さんのメッセージが契機になった（SNS 内部の）イベントが発生したことになる。内外の何れのイベントも会員のモチベーションによってメッセージの送信（投稿）が行われる。

イベント・ドリブンをノードとリンクのネットワークモデルに組み込めば、メッセージングに重点をおいた単純な構造とメカニズムをもったモデルができる。まずは、このイベント・ドリブン・モデルでイベントからメッセージが拡散することを計算し、その動態を SNS のメッセージングと比較してみる。

3.1.4　ノードとイベント、メッセージの共有の記述

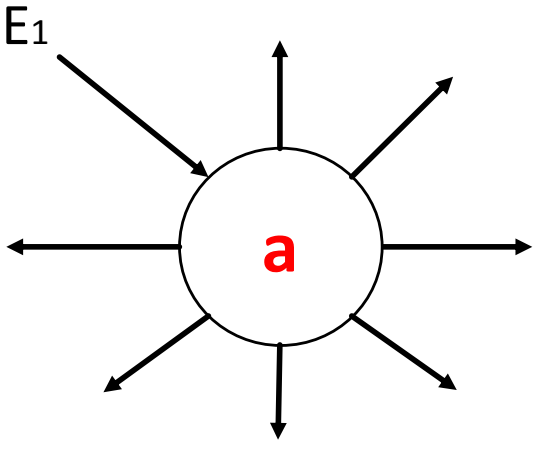


図 3-1　単一のノード

SNS をイベン・ドリブン・モデルで表現する際に、イベントやメッセージの共有はどう扱えばよいか。単純な１ノードから始めてみる。

単一ノード（a）は外部イベント E1 を知り、ノード(a)はメッセージを投稿したとすると、図 3-1 に示したように投稿したメッセージが他のノードにも共有されることは、メッセージをノード(a)のリンクにそって放出していることで示すことができる。リンク数が n 本であれば、１つのメッセージは n 個分のメッセージに複製されるように見える。

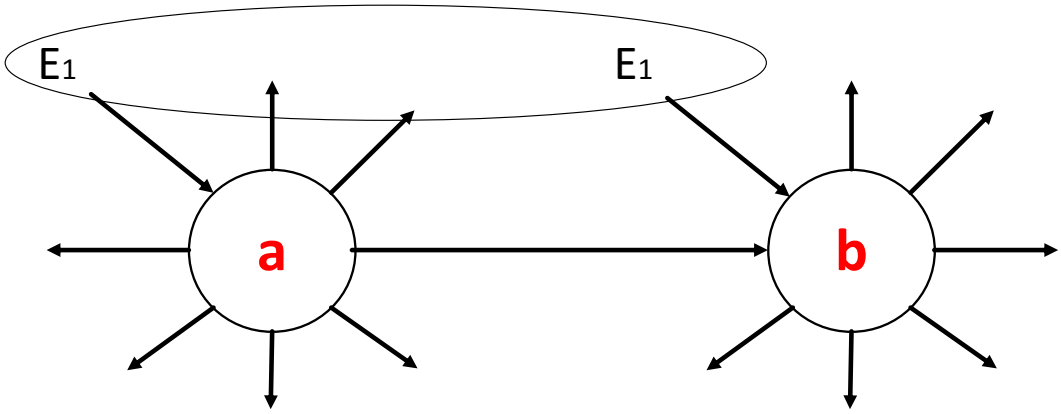


図 3-2 ノード（a）には n 個のリンクと信頼レベル tr があり、 ノード（b）は、ノード（a）とイベント E1 の両方から効果を受ける。

図 3-2 は、１つのリンクで繋がった２つのノード（a）と（b）を示している。ノード（b）は、ノード（a）とイベント E1 の両方からの効果を受けることを示している。ノード(b)では、ノード(a)の受け取ったイベント E1 と同じ情報を受け取ることができ、ノード(a)のメッセージとイベントのコンテンツ（内容）を比較できることを示している。メッセージは、ノード(a)のリンクを通じて、ノード(b)でもわかることから共有されていることがわかる。つまり、メッセージを共有するためにはノード間でリンクがなければならないことを示している。

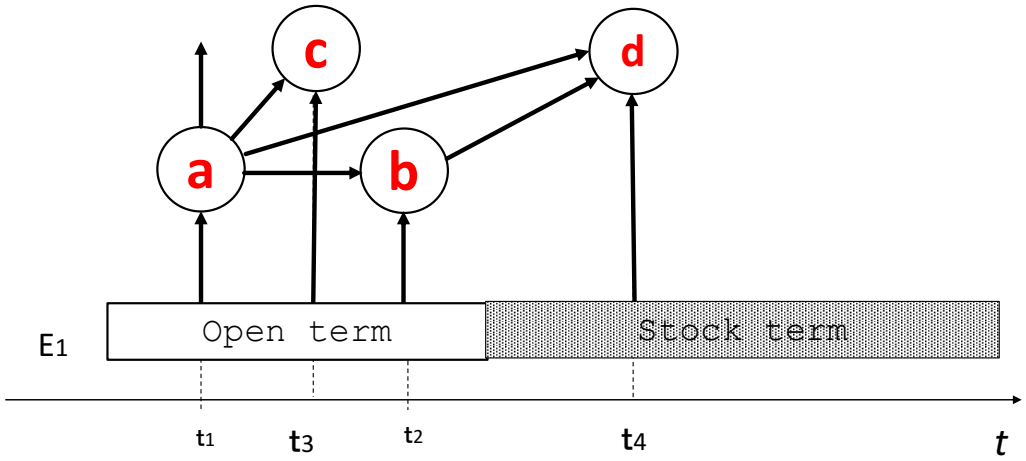


図 3-3 公開期間（open term）および蓄積期間(stock term)のあるイベント E を想定。 ノード（c）には E1 とノード（a）からのリンクがあり、ノード（d）にはノード（a）とノード（b）からのリンクがあり、ノード（b）はすでに E1 とノード（a）のリンクがある。

さらに、ノードの数を増やしてみよう。図 3-3 は、イベントが、一時的なものではなく、継続性をもった場合を示している。実際の SNS では映画の興行などが継続性のあるイベントであると言える[27]。例えば、映画が上映されている期間は公開期間として、どの時点でもメッセージの契機になりえる。映画の上映が終わっても、公開された事実はネット上に蓄積され、その間はメッセージの契機になりえる。

図 3-3 は、公開期間（open term）および蓄積期間(stock term)のあるイベント E1 を想定している。 ノード（c）には E1 とノード（a）からのリンク、ノード（d）にはノード（a）とノード（b）からのリンク、ノード（b）はすでに E1 とノード（a）のリンクがそれぞれ存在する。

図 3-2 に示した２つのノードの記述より複雑になっていく。イベントは継続しており、それを契機にする場合や他のノードのメッセージがイベントとなって、各ノードのリンクにそって、メッセージが時間とともに広がっていく。

ここまでで SNS の会員をノードとみなし、会員同士の繋がりをリンクで読み替え、イベント・ドリブンによってメッセージの発生（投稿、送信）とその拡散がモデル化できる準備ができた。つぎに、ヒューマン・コミュニケーションで重要なメメッセージと人間関係の影響を記述する。

3.1.5　コンテンツの信憑性と人間関係の信頼性の記述

ヒューマン・コミュニケーションの特徴である情報の「情」を示すパラメータをメッセージの内容（コンテンツ）と会員同士の人間関係に設定する。

図 3-4 ノード(a)はリンクを２つもち、時刻 t でイベントが発生して I(t)というメッセージ を受け取る。これを時刻 t+1 でパラメータ q, r tr に応じて処理をしてメッセージ I(t+1)を別のノードに送信する

１つのノード（a）に注目してみる。ノード(a)には図 3-4 に示すようにリンクを２つもち、時刻tでイベントが発生して$I(t)$というメッセージ を受け取ったとする。これを時刻$t+1$でパラメータ q、r と tr に応じて、適当な処理をしてメッセージ$I(t+1)$を別のノードに送信する場合を考えてみる。このとき、

$$I(t) = [q(t), r(t), tr(t)] \quad (1)$$

*I(t)*は、（1）式の３つのパラメータ$q(t), r(t), tr(t)$で決まるメッセージである。ここで、q（t）は、受信したメッセージの内容（コンテンツ）を肯定的または否定的な意見で変更および編集する量を示す。さらに、送信時の $q\ (t+1)$ は、モチベーションのパラメータ $M\_q\ (t)$ を伴って、関数 $f\ (.)$ で処理されるとする。

$$q(t\ +\ 1) = f\left(q(t)、M_{\_q(t)}\right) \quad (2)$$

信憑性 $r\ (t)$ は受信したメッセージの信憑性である。送信時の $r\ (t+1)$ は、モチベーションのパラメータ $M\_r\ (t)$ を伴って、関数 $g\ (.)$ で処理されるとする。

$$r(t\ +\ 1) =\ g\big(r(t)、M_{\_r(t)}\big) \tag{3}$$

初期（時刻0）の $tr\ (0)$ はメッセージを出した情報源（イベント $E1$ の情報提供者）から見た受信者の信頼性である。ここでは、イベントのニュースや広告を提供する側（メディア）は、受信者がメディアを信用してそのままメッセージを受け取ると仮定している。同様に $tr\ (t)$ は時刻tでノード（a）から見た受信者の信頼性である。

送信時の $tr\ (t+1)$ は、モチベーションのパラメータ $M\_tr\ (t)$ を伴って、関数 $h\ (.)$ で処理されるとする。

$$tr(t\ +\ 1) =\ h\big(tr(t)、M_{\_tr(t)}\big) \tag{4}$$

ここで、関数 $f\ (.)$ 、$g\ (.)$ 、および $h\ (.)$ は、$q\ (t)$ 、$r\ (t)$ 、および $tr\ (t)$ の編集関数と呼ぶことにする。 関数 $f\ (.)$ 、$g\ (.)$ 、および $h\ (.)$ は、モチベーションのパラメータである $M\_q\ (t)$ 、$M\_r\ (t)$ 、および $M\_tr\ (t)$ の影響を受ける、モチベーションのパラメータ $M\_q\ (t)$ は、コンテンツの変化量を示すスケールファクターである。 $M\_r\ (t)$ と $M\_tr\ (t)$ は、コンテンツの信憑性と情報源の信頼性のレベルである。

後述のシミュレーションでは、関数 $f\ (.)$ 、$g\ (.)$ 、および $h\ (.)$ つぎのように簡素化して考察する。

ここで(1)式から(4)式の離散方程式の時間微分を考え、先行研究の２.１.４で紹介した拡散方程式(a)を用いてみる。

$$\frac{dI_i(t)}{dt} = C_iA(t) + \sum_j B_{ij}I_j(t) + \sum_j B_{ij}I_j(t)\sum_k D_{jk}I_{jk}(t) \tag{a}$$

(a)式で単一ノードなのでkの項を除き、外部からイベントの入力項$A(t)$がない場合と考えられることから、(a)式は、拡散速度の重みmを乗じた次式に単純化できる。

$$m\frac{dI(t+1)}{dt} = \sum B * I(t) \tag{5}$$

さらに(5)式で$M_q(t)$, $M\_t(t)$, $M_tr(t)$の定義から比較すると、mは $1/M\_q\ (t)$ 、Bが $M_r(t)*M_\mathrm{tr}(t)$と置換できることがわかる。(5)式で、$m \geq 1$であり、$B = 1$であるならtに対する指数関数となる。つまり、編集が 100%以上でコンテンツの信憑性と受信者の信頼性がともに 100%なら、指数的に増加傾向を示すはずである。

３.１.６　メッセージの改変と編集

前節のメッセージの変更と編集の定義については、関数 f（.）、g（.）、および h（.）と、q（t）、r（t）、および tr（t）の編集関数という抽象的な記述が入ってきたので、(2)'式から(4)'式で示したように「編集」の意味付けを示す：

ここで、図 3-4 のノード(a)について

① 編集せずに投稿する場合は、$M\_q\ (t) = 1$

② 投稿するメッセージの内容（コンテンツ）を変更する場合は、$|M\_q\ (t)| < 1$ で、
- 正または負の極性は変更にせずに編集する（内容を膨らませる）。
- 極性を変えて編集する（内容を真逆にして膨らませる）

③ 投稿しない（コンテンツが許容範囲（しきい値）を超えているは、$M\_q\ (t) = 0$

が考えられる。ここでいう極性とは、コンテンツを肯定する場合を正、コンテンツを否定する場合を負で表した。

3.1.7　メッセージングに対する戦略

図 3-4 の１つのノードに対して、リンクが複数あり、他のノードから同時にメッセージを受信する場合もある。そこで同時にメッセージを受ける場合、メッセージは次の戦略で設定されていると仮定することにした。

① 各ノードには、M_q（t）、M_r（t）、および M_tr（t）のしきい値範囲が個別にあるとする。
② メッセージは、M_q（t）、M_r（t）、および M_tr（t）の最大値を示すメッセージを選択する。
③ モチベーションが高すぎるメッセージや受信した情報の信憑性や人間関係の信頼度が極端に低い場合はメッセージが送信されない。

もちろん、これ以外の戦略で設定することも可能である。今回とった戦略について説明すると、①は SNS の会員の特性に偏りがない設定とし、②は、メッセージが各ノードのモチベーションによる編集が大きく、コンテンツの信憑性や人間関係の信頼度も高いものを選択することから、メッセージの拡散がモチベーションによる影響が大きくなるように選んでいると言える。③は②と同様、メッセージが健全に拡散するのを助けるために、極端なメッセージの編集が行われた信憑性の低いメッセージや信頼がおけない情報源からのメッセージを拡散しない設定とした。

次節のシミュレーションでは、上記の戦略で１つの個人的な動機からメッセージが拡散する様子を再現することにした。

図 3-3 の例で考えてみよう。

t = t1 時点で、ノード(a)は、公開期間にあるイベント E1 の広告などの情報を得たとする。

- ノード(a)は、E1 の情報をもとに、投稿するモチベーションがあり、例えば、*M_q（t1）＝1* で、E1 の情報を編集も極性の反転も行わないというモチベーションを持っている。
- イベント E1 のコンテンツは、大手広告会社の広告で一応信憑性は高いと判断して、*M_r（t1）＝1* とノード(a)は判断する。
- イベント E1 との人間関係は、大手広告会社の広告の受信者は一応広く信頼されていると判断して、*M_tr（t1）＝1* とする。
- ノード(a)のモチベーションのパラメータである M_q（t1）、M_r（t1）、および M_tr（t1）を関数 f（.）、g（.）および h（.）に処理させ、t = t1 でメッセージを発生、SNS に投稿する。実際の SNS では関数 f（.）、g（.）および h（.）は各ノードに異なることになるが、ここでは簡素化して、各ノードで関数 f（.）、g（.）および h（.）は同じで、パラメータ M_q（t）、M_r（t）、および M_tr（t）が変数として扱う。

t = t2 時点で、ノード(b)は、リンクのあるノード(a)からのメッセージを閲覧でき、さらに公開期間にあるイベント E1 の広告情報を直接みることもできる。ノード(a)のモチベーションのパラメータである M_q（t2）、M_r（t2）、および M_tr（t2）は、３.１.７の戦略に従って、ノード(a)からのメッセージとイベント E1 の広告情報の中で最大になるようなメッセージまたはイベント情報を選択する。選択したメッセージまたはイベント情報の M_q（t2）、M_r（t2）、および M_tr（t2）をノード(b)

に対して関数ｆ（.）、ｇ（.）およびｈ（.）に処理させ、t = t2 でメッセージを発生、SNS に投稿する。

- t = t3 時点で、ノード(c)も上記のノード(b)と同様に、リンクのあるノード(a)からのメッセージを閲覧でき、さらに公開期間にあるイベント E1 の広告情報を直接みることもできる。上記と同様な流れで、t = t3 でメッセージを発生、SNS に投稿する。
- t = t4 時点で、ノード(d)も上記のノード(b)、(c)とは異なり、リンクのあるノード(a)と(b)からのメッセージの両方を閲覧でき、さらに公開期間ではない蓄積期間のイベント E1 の広告情報を直接みることになる。公開期間と蓄積期間の広告情報ではコンテンツは同様であるが、公開前後で人間のコミュニケーションでのモチベーションに変化すると考えられる[17]。つまり、ノード(d)の M_q（t4）、M_r（t4）、および M_tr（t4）でコンテンツの信憑性は維持されるが、過去の出来事に対するメッセージの編集が行われる。例えば、すでに終わったイベントに対する評価や感想が追加される可能性が出てくる。
- リンクのあるノード(a)と(b)からのメッセージの両方を閲覧するとき、ノード(d)は、コンテンツの内容がノード(a)と(b)の何れも信憑性のあるものであっても、人間関係の信頼度が異なる場合、３.１.７の戦略に従って、信頼性が高いメッセージを選択することになる。

３.２　１ノードからの情報の拡散

前節までの仮定を置いて、情報理論に基づき、情報源が１つで、いくつかの送受信が行われるノードで構成された図 ３-3 のようなネットワークを考える。情報の送受信を任意に設定できるランダムネットワークの構成をとる。

３.２.１　関数ｆ（.）、ｇ（.）およびｈ（.）の記述

シミュレーションによる計算に先立って、編集関数ｆ（.）、ｇ（.）およびｈ（.）の実装を考える。前節で示したように各ノードで受信したメッセージは関数ｆ（.）、ｇ（.）およびｈ（.）で投稿するメッセージに変換される。これら編集関数は前節の仮定と同様に全てのノードで同じであるとすると、モチベーションによる変換後のメッセージの変化は、パラメータ M_q（t）、M_r（t）、および M_tr（t）の変化で記述することになる。

ここで、(2)式から(4)式で以下のように簡素化して考える。

$$f(t) = q(t) * M_{\_q(t)} \qquad (2)'$$

$$g(t) = r(t) * M_{\_r(t)} \qquad (3)'$$

$$h(t) = tr(t) * M_{\_tr(t)} \qquad (4)'$$

(2)'式は、入力されたメッセージ *I(t)*のコンテンツ *q(t)*を *M_q（t）* 倍するもので、*M_q（t）＝1* ならばコンテンツを変えずに時刻t＋1で出力する。*M_q（t）＜1* ならば、コンテンツを減らし、*M_q（t）＞1* であれば、コンテンツを増やす編集を行うことを意味する。*M_q（t）＝0* は、コンテンツを時刻 t+1 で出力しないことを意味する。

(3)'式は、コンテンツの信憑性で$0 \leq M_r（t）\leq 1$とした。全く信憑性がない場合は 0、完全に信憑性がある場合は 1 とし、*r(t)*を *M_r（t）* 倍する。

(4)'式は、の信用性で$0 \leq M_tr（t）\leq 1$とした。受信者が全く信用できない場合は 0、完全に信用している場合は 1 とし、*tr(t)*を *M_tr（t）* 倍する。

３つの編集関数をプログラムで実装し、各パラメータ M_q（t）、M_r（t）、および M_tr（t）を変えて、各時刻の情報の広がりを観測する。[35]

3.2.2　初期条件

次の計算では、各ノードと初期イベントに多くの初期条件を設定する必要がある。初期条件の値とスイッチフラグは次のとおりである。[35]　スイッチフラグとは、例えば、指定した M_q（t）で値をランダムにするか、固定値にするかを実験者が指定できるスイッチである。

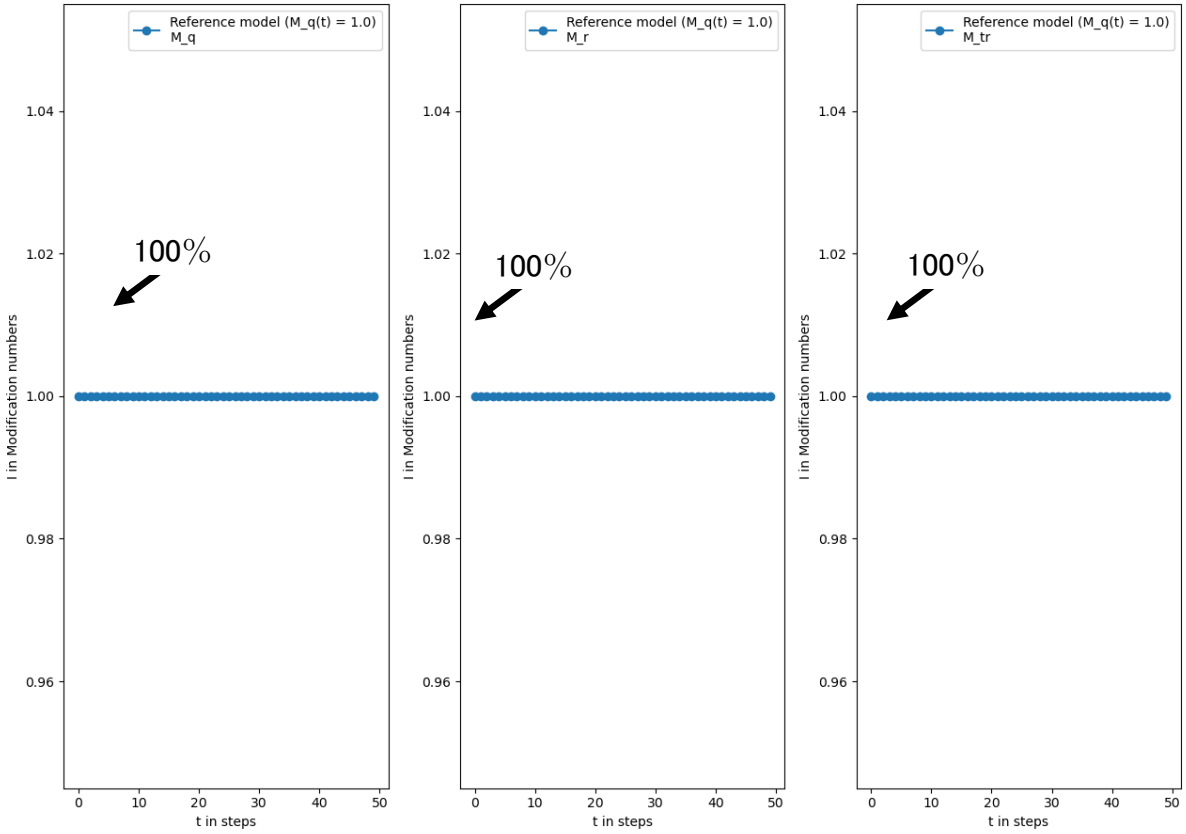
各パラメータは(1)から(4)式に対応する：

- st：実験の回数（反復、ステップ）：通常 st = 50
  このシミュレーターは、初期条件から繰り返し実行できる設計で、ランダムな変数を含む場合、繰り返しを増やすことで、結果の傾向が読み取り易くなる。
- I（0）：初期メッセージ：I（0）には、q（0）、r（0）、および tr（0）が含まれる。最初にノードにリンクあるいは外部イベントとして受信されるメッセージを示す。
- m（0）：変更係数：m（0）には、M_q（0）、M_r（0）、および M_Tr（0）が含まる。変更係数は、コンテンツを変更する度合いを表す。
- q_max：Q（t）の制限サイズ：通常 q_max = 50,00。コンテンツの編集をどこまで許容するかを示す最大値。編集は、コンテンツの極性など内容の変更や長さの変更を含む。この値は、受信したコンテンツが編集の度合いが大きく、長い場合に「極端なメッセージ」と判断して受信したノードが送信をしない設定である。SNS で「極端なメッセージ」とは、ヒューマン・コミュニケーションの中で倫理性を欠けるものや誹謗中傷などを指す。[9]
- m_max（0）：M_q（0）、M_r（0）、および M_tr（0）の最大サイズ。の係数は、M_q（t）、M_r（t）、および M_Tr（t）の各ランダムスイッチが「オン」の場合に有効になる。 ランダムな M_q（t）、M_r（t）、M_tr（t）でコンテンツが「極端なメッセージ」になることを防止する制限である。
- m_th（0）：M_q（0）、M_r（0）、および M_tr（0）のしきい値。しきい値は、この値を超えると、メッセージを送信しない設定である。最大値は、SNS 全体での制限であるが、しきい値は各ノードが個別に持つ制限である。
  なお、M_q（0）、M_r（0）、および M_tr（0）のしきい値のランダム化は、M_q（0）、M_r（0）、および M_tr（0）のそれぞれのしきい値を 0%から 100%までの疑似乱数を乗じて生成している。詳細は３.２.３で説明する。
- random_sw：M_q（t）、M_r（t）、M_Tr（t）のスイッチフラグ。
  M_q（t）、M_r（t）、M_tr（t）のしきい値：これらは、極性の正と負の反応に関わらず各ノードを３.２.３のパーソナライズをするためのスイッチ。
  なお、M_q（t）、M_r（t）、M_Tr（t）のランダム化は、いずれも 0.0 から 1.0 までの浮動小数点型疑似乱数で生成している。
- Mq_pn_sw：極性の正と負に反応するかどうかのスイッチフラグ。

これらの初期条件や各種パラメータを使う付録１：第３章のシミュレーションに掲載する。

3.2.3　モデルの検証

前節までに準備したイベント・ドリブン・モデルを自明な条件で計算を行い、モデルを検証することにした。



(a) q(t)　　(b) r(t)　　(c) tr(t)

図　3-5 シミュレーションの自明な参照モデルとして編集や変更がない条件での結果。 図の(a)、(b)および(c)の動機付けファクタは一定であることがわかる.

M_q(0）、M_r（0）、および M_Tr（0）を全て 100%として計算した結果が図 3-5 である。時間がたっても、一定のレベルで拡散を続けて平坦なグラフになっていることがわかる。

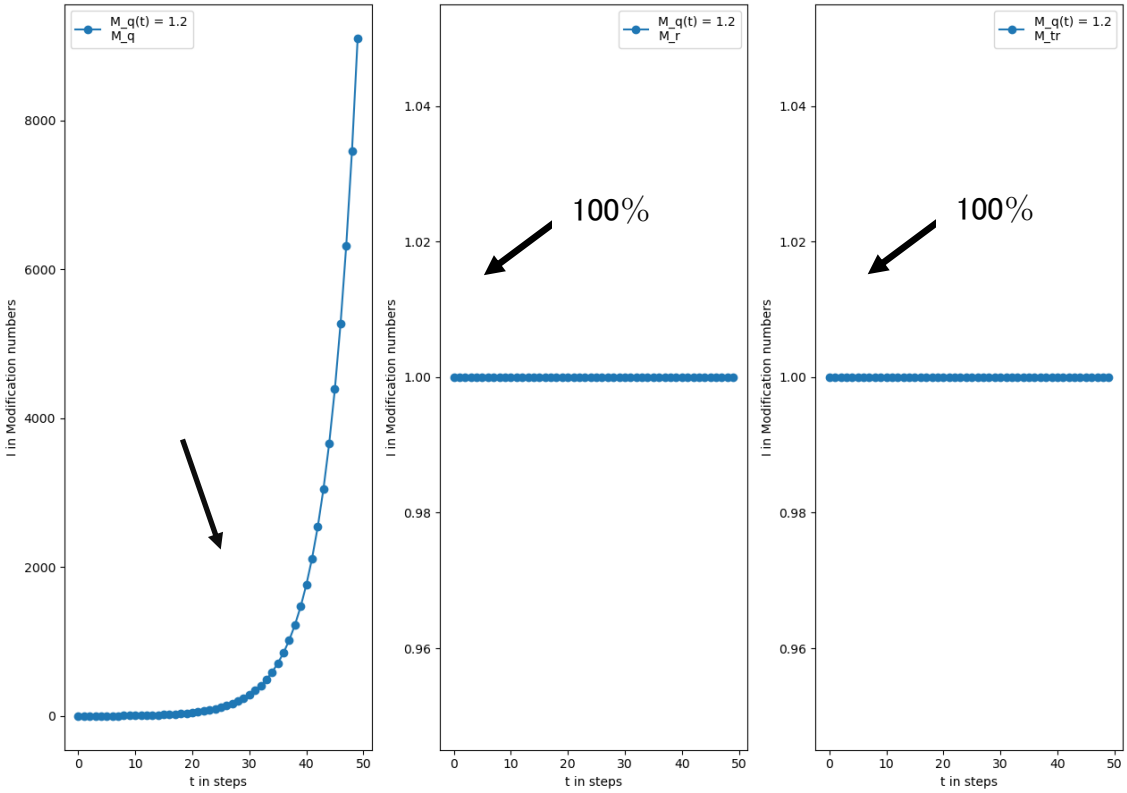
3．1．7 メッセージ戦略の②「M_q（t）、M_r（t）、および M_tr（t）の最大値を示すメッセージを選択する」に則り、メッセージは改変されることなく観測時間の全てにわたって伝搬され拡散した。

次に、M_q(t）を変え、メッセージの内容（コンテンツ）を 100%以上で改変編集し、信憑性や信頼性が 100%であれば、コンテンツを膨らましたものであっても、他のノード（会員）に伝わっていくことが予想できる。100%の信憑性と 100%の信頼性レベルで 50 ステップの間に M_q（t）= 1.2 の結果を図 3-6 に示した。q(t)は、予想通り指数関数的な増加を示し、メッセージ戦略②に従ってメッセージは拡散した。

SNS では、最初 1 会員が、メッセージを別の会員に伝えるが、メッセージを受信した会員が編集を加えて、コンテンツを膨らませ、さらに繋がっていく。

これは明らかに、噂話のように連鎖的な変更が雪だるま式で情報が拡散する現象である。

以上の自明な条件の設定でコンテンツの拡散モデル(5)式を確認し、導入したモデルによるシミュレーションの検証を確認した。

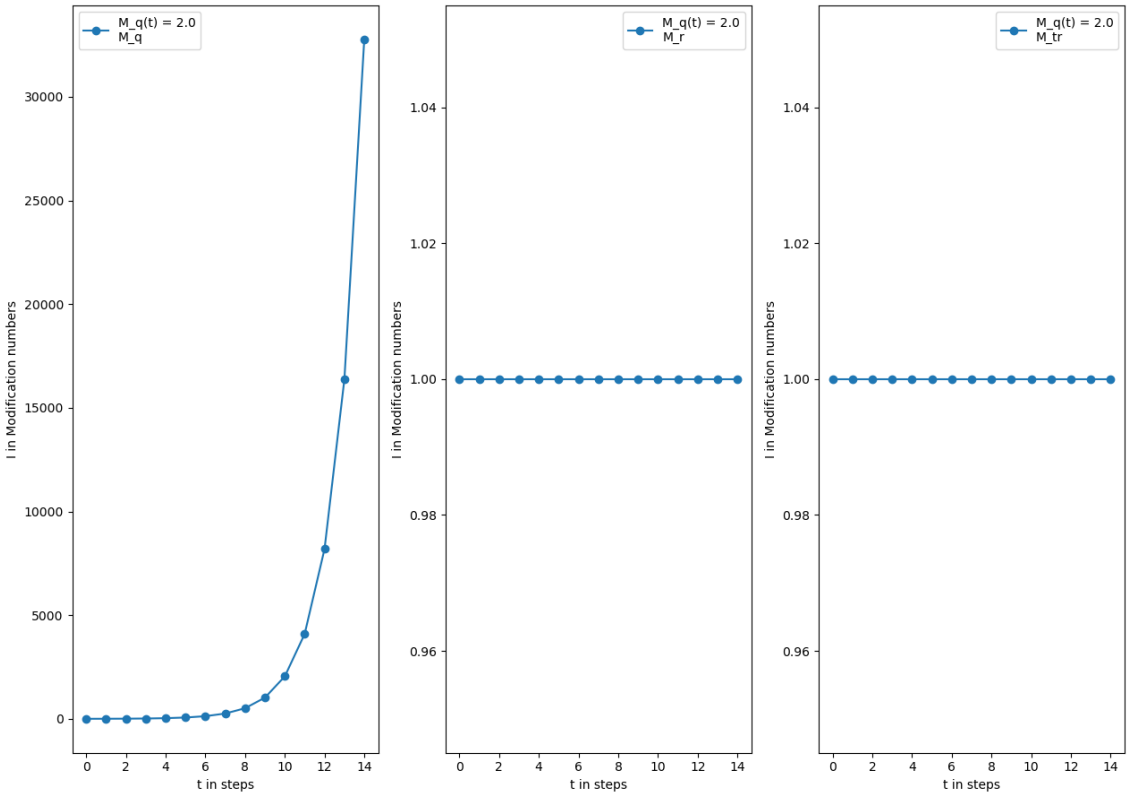


(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-6　q(t)は指数関数的増加を示す。(M_q (t) = 1.2)

3.2.1　M_q（t）の影響

モチベーションのダイナミクスをさらに観察するために、100%の信憑性と 100%の信頼性のままで M_q（t）= 2.0 を設定した結果が図 3-7 である。



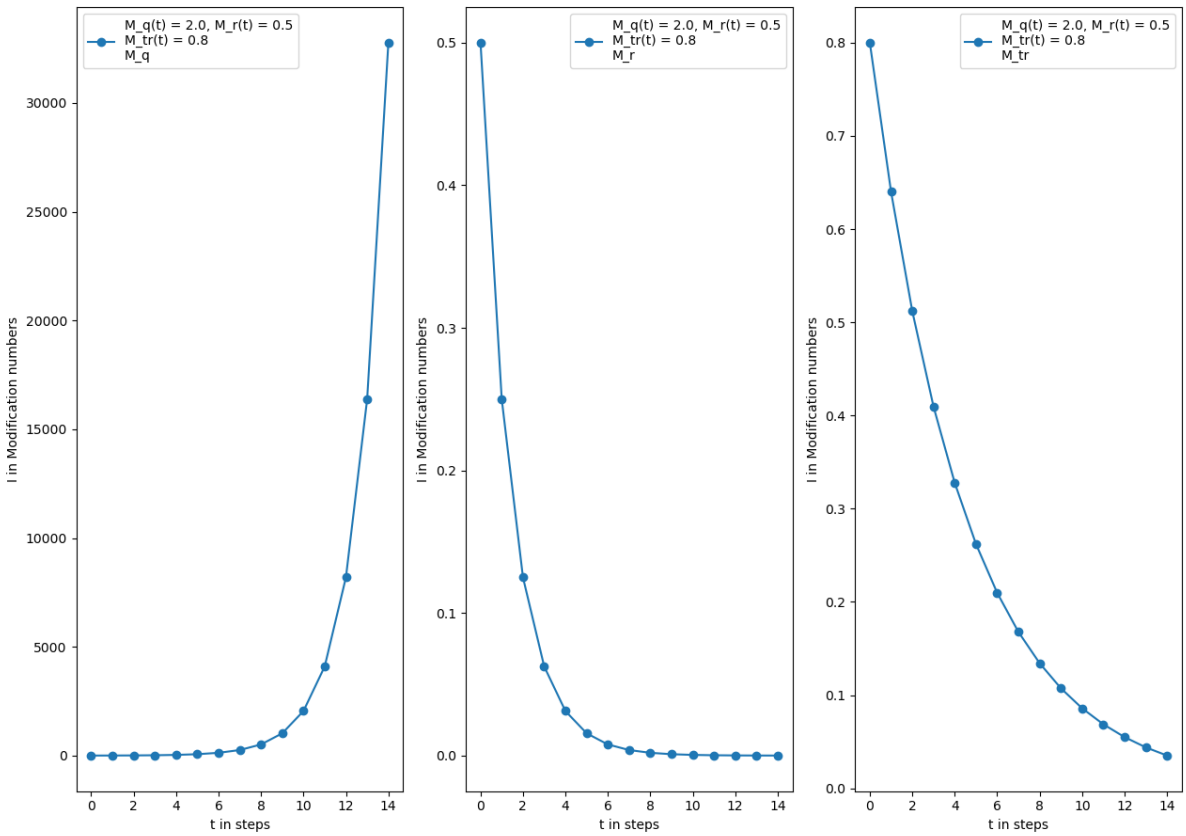
(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-7-100%の信憑性と 100%の信頼レベルのままで M_q（t）= 2.0 を設定

図 3-6 と図 3-7 を比較すると図 3-7 の方が、q(t)の増加が大きいことがわかる。つまり、M_r(t)や M_tr(t)が 100%であるため r(t)、tr(t)も一定で、最大値の判定は q(t)で決定され、これによってメッセージが発信される。q(t)が大きいことは、短い時間（ステップ数が少ない間）でメッセージが発信されることになり、観測時間全般にわたってメッセージの拡散が速いことになる。コンテンツがノードを通る間に修正が連続的に行われることで、メッセージの拡散速度が大きくなる。コンテンツの修正や編集が連続的行われることは、メッセージを広げることに効果があることがわかる。言い換えれば、修正や編集が行われるようなメッセージであればあるほど、会員の興味を引き続けるとも言える。

3.2.2　M_q(t)、M_r(t)および M_tr(t)の変動

M_q(t)の変動に続いて、M_r(t)、M_tr(t)を次に変動させてみよう。M_r(t) = 50%、M_tr(t) = 80%の結果が図 3-8 である。



(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-8　M_r(t) = 50%, M_tr(t) = 80%

M_r(t) = 50%、M_tr(t) = 80%では図 3-7 と図 3-8 を比較して、先ずは両図の(a)の q(t)は同じであるが、両図の(b)の r(t)と(c)の tr(t)を比較すると、図 3-8 はともに急激に減少し、t = 10 前後の r(t)で 0 となり、メッセージへの戦略③の「モチベーションが高すぎるメッセージや受信した情報の信憑性や人間関係の信頼度が極端に低い場合はメッセージが送信されない。」に則り、メッセージの拡散は停止することが観測できた。M_r(t)と M_tr(t)の両方が指数関数的に減少する。

これは、最初のイベントからの元の情報に、送信中の信憑性と信頼性が低いために多くの編集による変更が含まれ、メッセージの信憑性が低下し拡散しなくなる。

3.2.3　パーソナライズ（レベル 1）

ここまでの計算は、SNS 内部のすべての会員（ノード）が実際のコミュニケーションのように機能してするわけではない。まだまだ、エラーやトラブルを起こすネットワーク機器間の通信の状況を表したものに近い。

そこで、実際の SNS のように各会員が個性を持つような仕組みをモデルに組み込むことを考える。各ノードは人間の行動として独立して機能する仕組みを取り込むことになる。

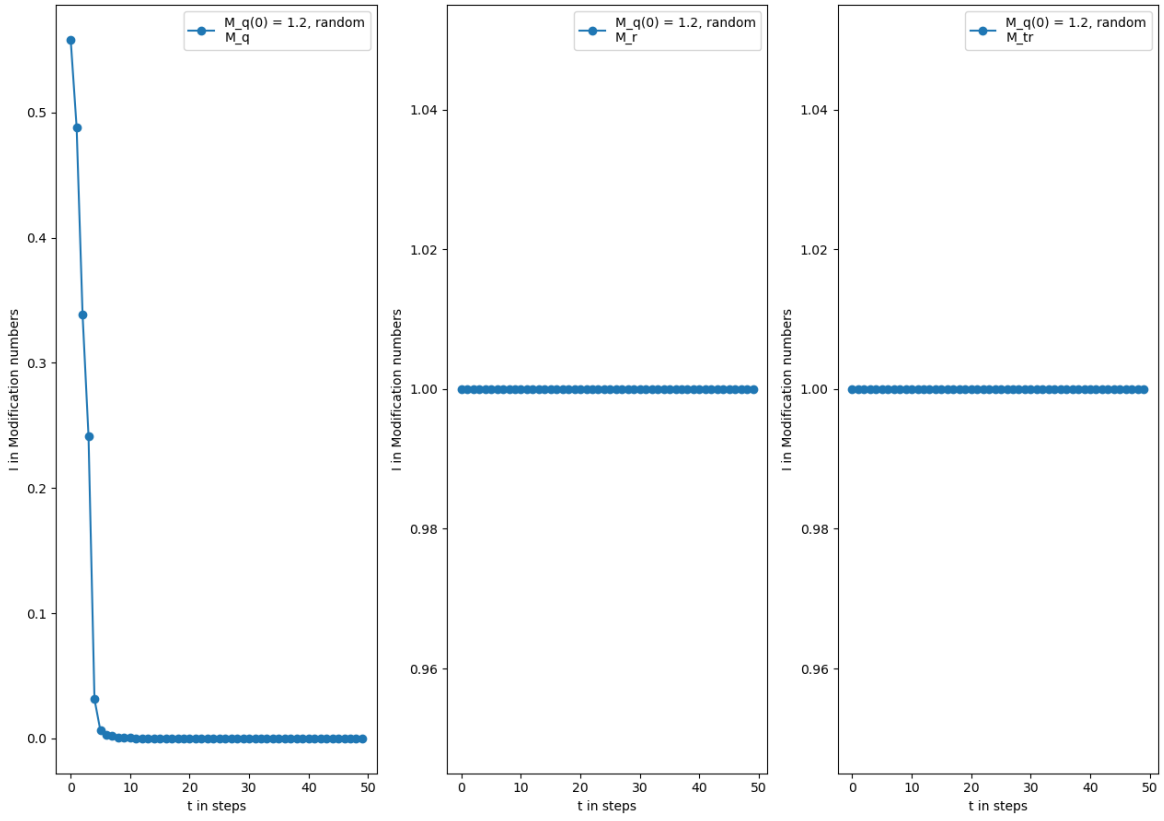
例えば、「サッカーチーム X が 5 対 3 で勝利！」というメッセージを会員 a が受けたとする。M_q（t）= 2.0 は、このメッセージを「2021 年 9 月〇日横浜スタジアムで行われた J リーグのナイトゲームで・・・・チーム X が 5 対 3 で勝利した！」とし

て編集修正したメッセージに変換することに相当する。このメッセージを受信した会員 b は、

- 2021 年 9 月〇日にチーム X が５対３で勝利した！
- ナイトゲームでチーム X が５対３で勝利した！
- チーム X が５対３で勝利！

のように、いくつかのバリエーションの中から興味のあるメッセージを利用することだろう。会員 b のモチベーションを高め、上記の想定されるメッセージのどれを選択するかは会員で同じではなく異なることがわかる。このような選択の違いをシミュレーションでは M_q（t）をランダムな変数として、それ以外の、M_r（t）と M_tr（t）は定数とする。このようにノードを SNS の会員に近づけることを「パーソナライズ」と呼び、レベル１では M_q（t）をランダムな変数とする。

レベル 1 の結果の一部を図 3-9、図 3-10、および図 3-11 に示す。結果の一部としたのは、M_q（t）をランダムな変数としたためであり、全てを掲載できないためである。



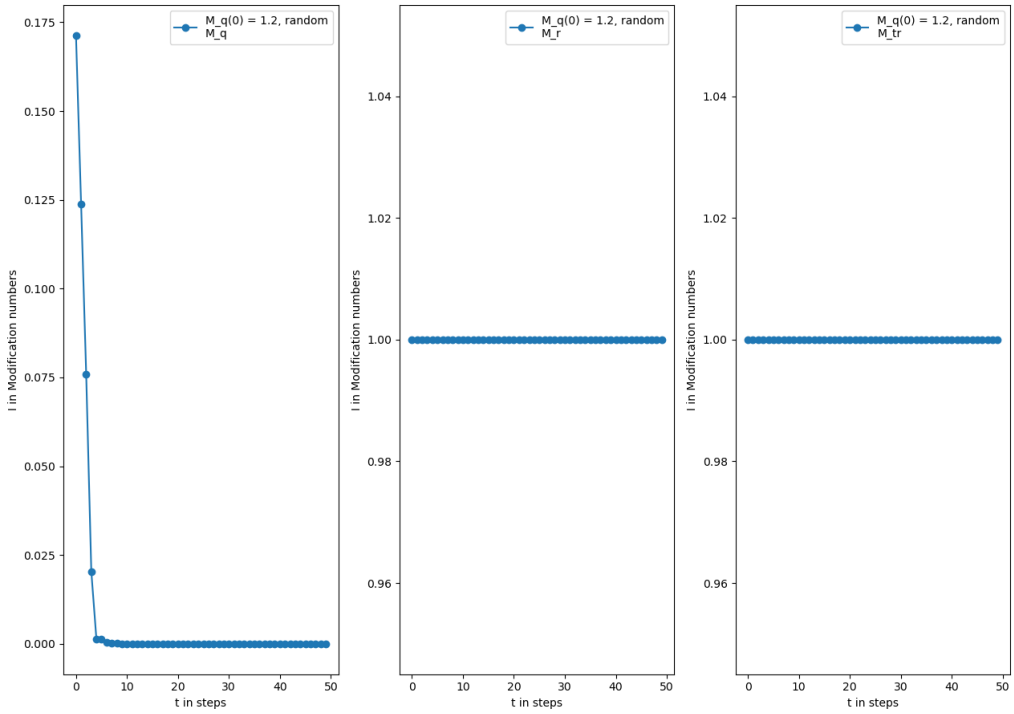
(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-9 初期値 M_q(0) = 1.2 として M_q（t）をランダム化した場合の結果の例

結果の図 3-9 から図 3-11 までみると、M_q（t）が、図 3-5 の結果よりも急速に減少し、最大値が 100%の信憑性と 100%の信頼性でランダムになっていることを示している。

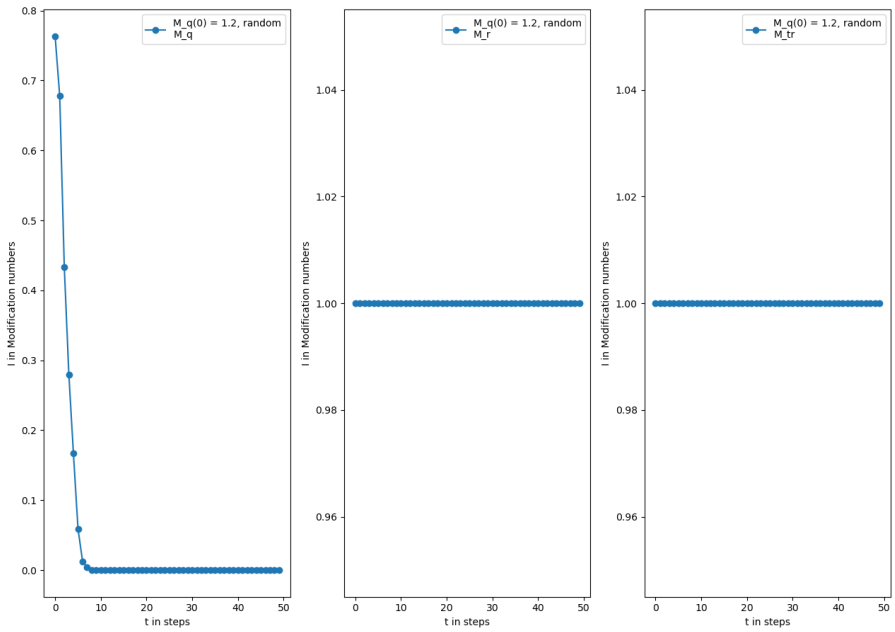
これは、SNS の会員にとって、外部のニュースやイベント情報の拡散が意外にも急速に拡大しない可能性があることを意味している。

次に、ランダムな M_r（t）と M_tr（t）にした場合の結果を図 3-12 に示す。ランダムな M_q（t）のみよりも、元の情報の拡散がより迅速に終了する。



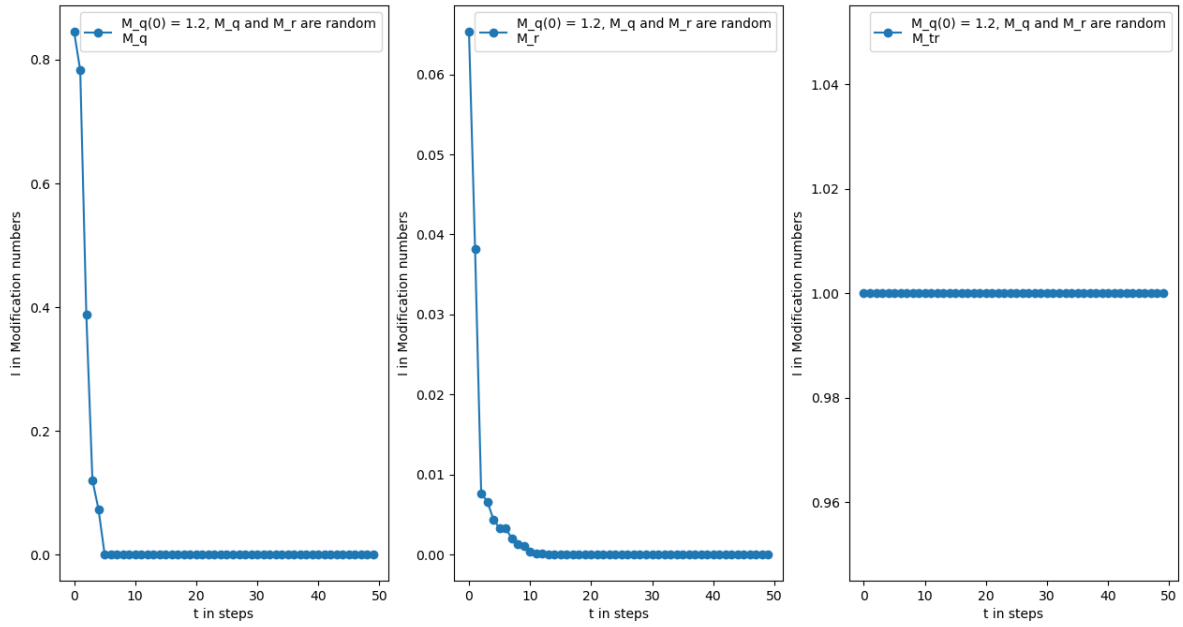
(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-10 初期値 M_q(0) = 1.2 として M_q（t）をランダム化した場合の結果の例



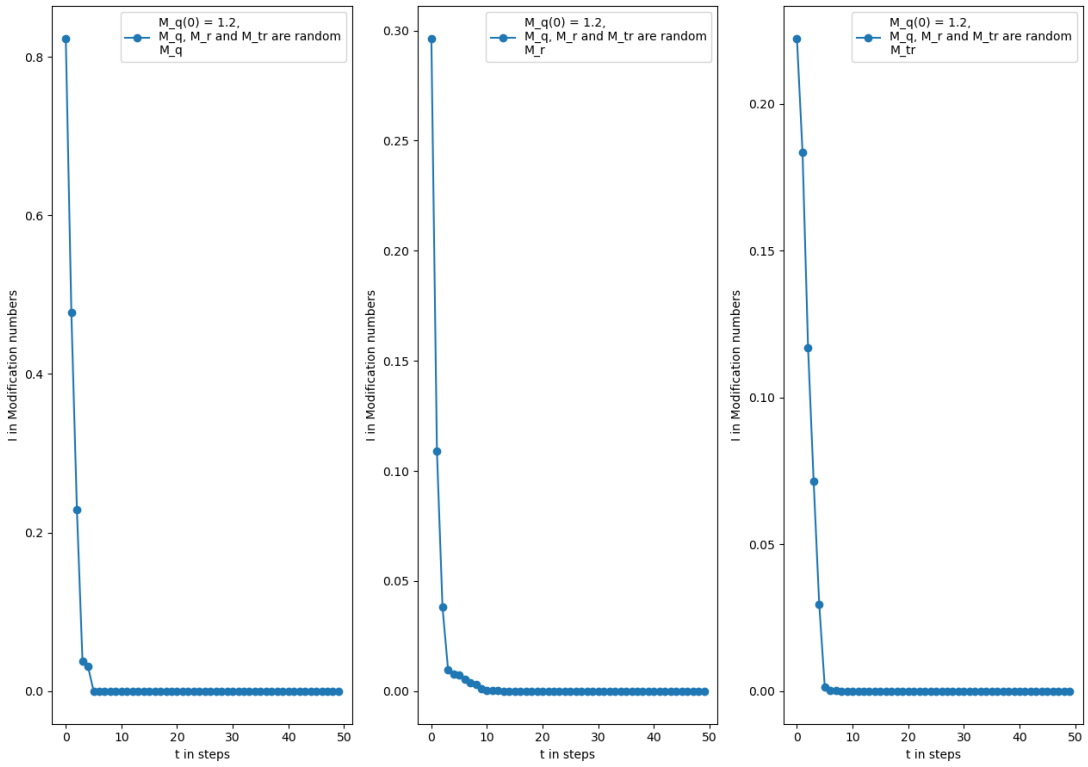
(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-11　初期値 M_q(0) = 1.2 として M_q（t）をランダム化した場合の結果の例



(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-12 初期値 M_q(0) = 1.2 として M_q (t) と M_r (t) をランダム化した場合の結果の例



(a) q(t)　　(b) r(t)　　(c) tr(t)

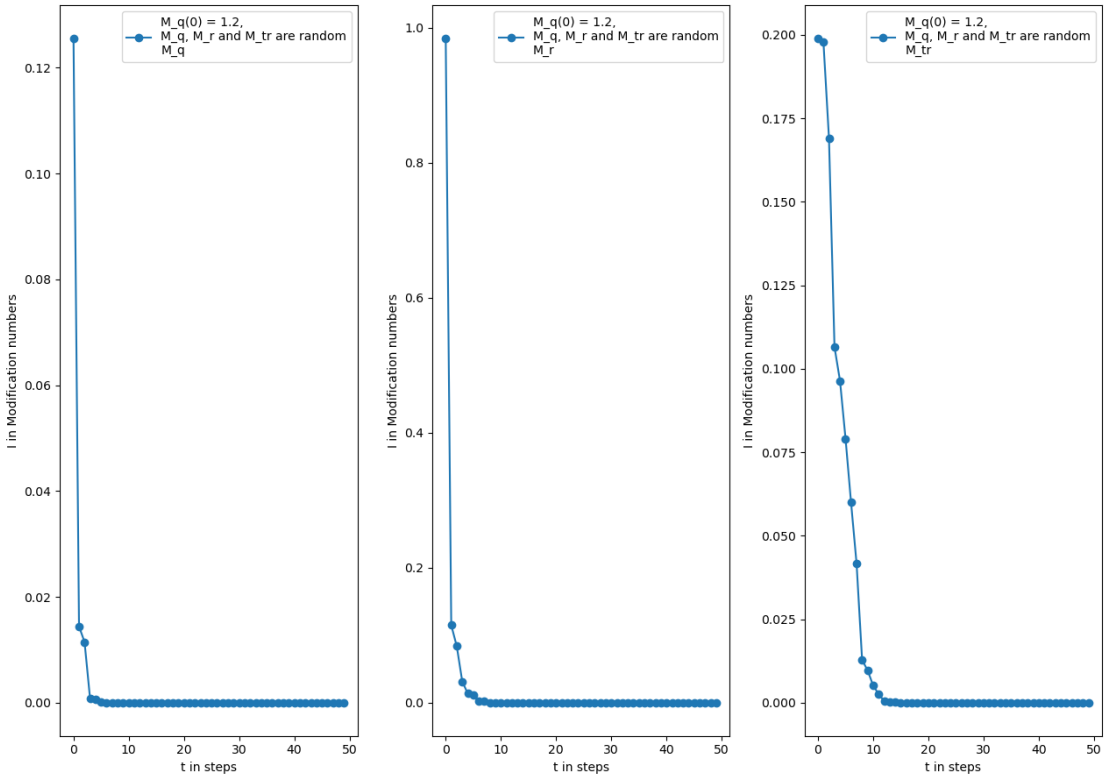
図 3-13 初期値 M_q(0) = 1.2 で M_q(t), M_r(t) および M_tr(t)をランダム変数にした結果の一部

M_q（t）、M_r（t）および M_tr（t）を全てランダム変数にした結果の一部を図 3-13 から図 3-15 までに示す。どの結果もメッセージの拡散の停止は、早期に発生している。コンテンツの信憑性、人間関係の信頼性も高くなければ、メッセージは拡散しないことがわかる。この条件は、シミュレーションのメッセージの終結状態を表示させてみると、各ノードのしきい値が、メッセージを終結させるように働くことでわかる。つまり、コンテンツに信憑性の疑いがあっても、人間関係で信頼性が高いとメッセージは拡散し易くなると言える。SNS でいう、知人や信頼性の高いと思われる情報源のメッセージを鵜呑みにすることにあたる。逆に、コンテンツの信憑性が高くても、人間関係の信頼性が低いと、メッセージが拡散しにくい。いくら言っていることが正当であっても、胡散臭い人間の話は聞かないといった態度である。

以上をまとめると、

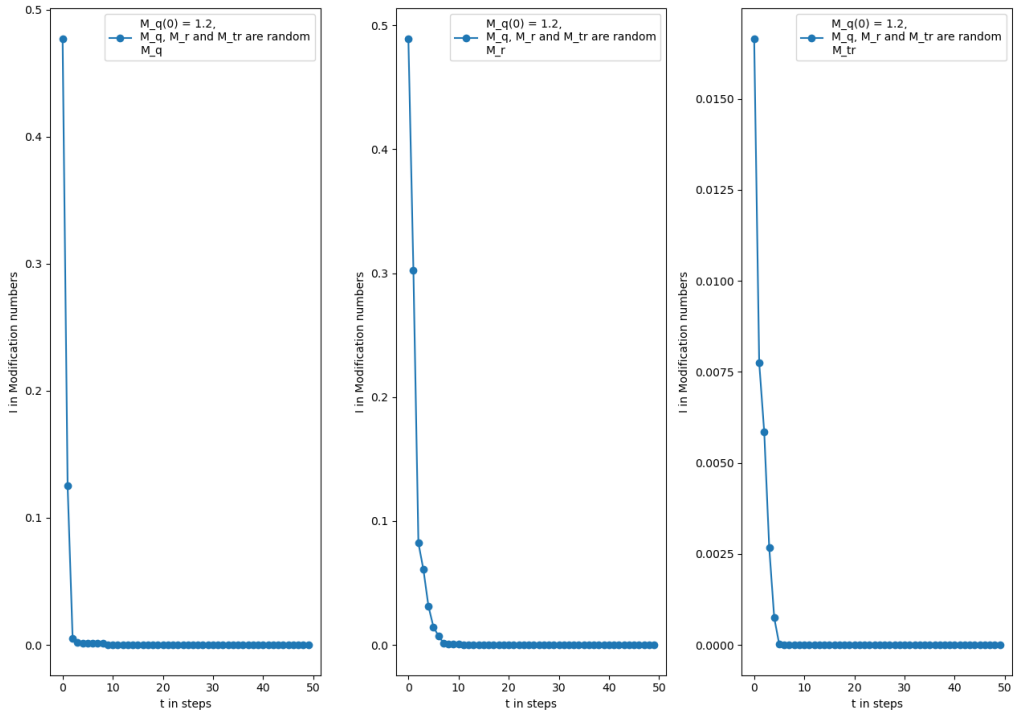
① 元のメッセージの拡散は、定数の M_q（t）、M_r（t）、および M_tr（t）よりも急速に終了する

② 早期終了では、定数 M_q（t）よりも元のイベントからの情報量が少なくなることとなる。より広い拡散が必要な場合で、かつ内容を誇張しないためには、M_q（t）= 1.0 のレベルを維持する必要があることを意味する。 これは、実際の SNS のインフルエンサーまたはスポークスパーソンである会員が M_q（t）= 1.0 を維持することで、ありのままにメッセージを加筆修正なしで伝える必要があることを意味している。



(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-14 初期値 M_q(0) = 1.2 で M_q(t), M_r(t) および M_tr(t)をランダム変数にした結果の一部



(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-15 初期値 M_q(0) = 1.2 で M_q(t), M_r(t) および M_tr(t)をランダム変数にした結果の一部

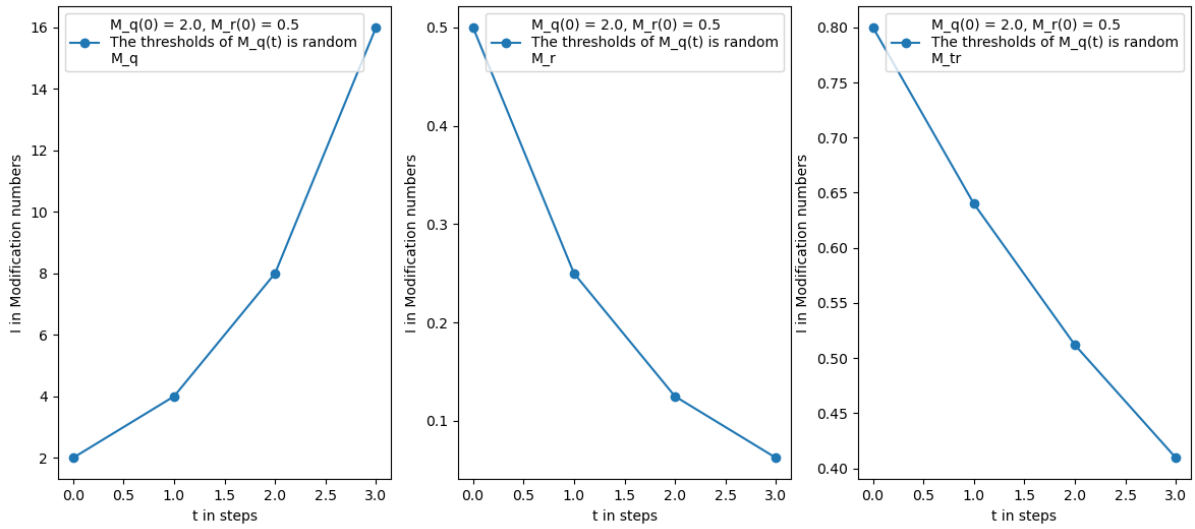
3.2.4 パーソナライズ（レベル 2)

パーソナライズのレベル２では、M_q（t）、M_r（t）、および M_tr（t）のしきい値を変動させる。 変動によるバリエーションは、会員の個性のファクタと考えることもできる。 図 3-16 から図 3-18 の結果は、コンテンツの（編集度合いの）しきい値を変動させ、図 3-19 と図 3-20 はコンテンツと人間関係の両方のしきい値を変動させている。

これらの結果は、M_q（t）= 2.0 にもかかわらず、各ノードのしきい値が固定されている３．２．３よりも短い時間で拡散が終了することを示している（図 3-6 を参照）。

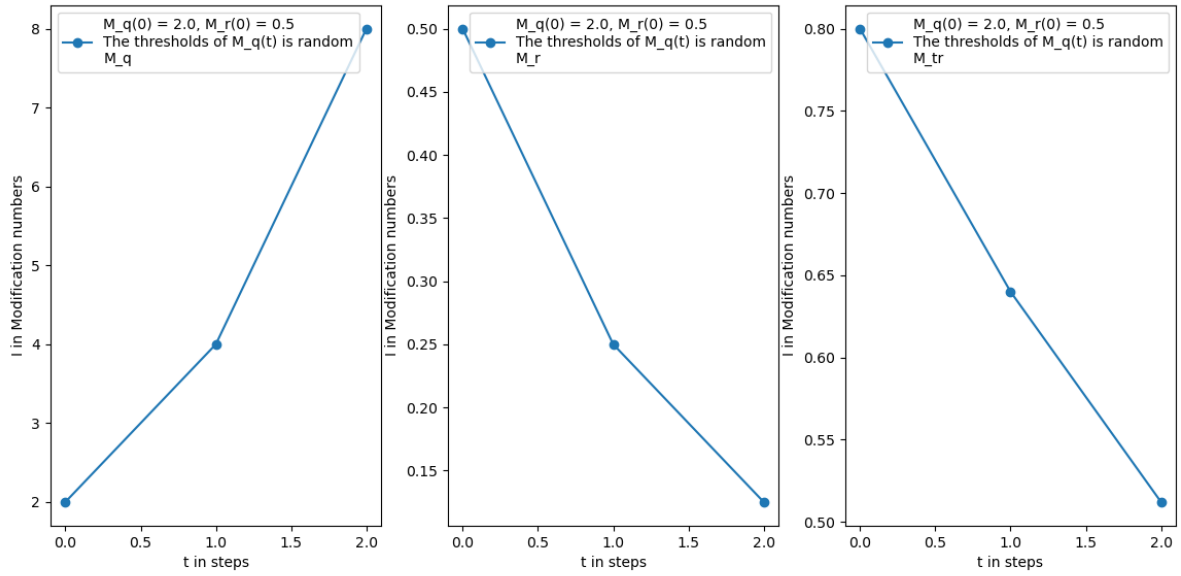
M_q（t）= 2.0 は、図 3-6 では単純に拡散していくが、しきい値を変動させることは、ノードのメッセージの拡散速度へのブレーキとして機能している。 この状況は、SNS において、個性が際立って異なっている中で、メッセージを拡散させることが難しいことを示している。

また、図 3-17 と図 3-18 の違いは、同じ最初のメッセージが拡散した後、各ノードのしきい値の変動で、いくつかのバリエーションが生まれることを示している。これは、偶然性に左右されて、メッセージの拡散がおこることを意味している。



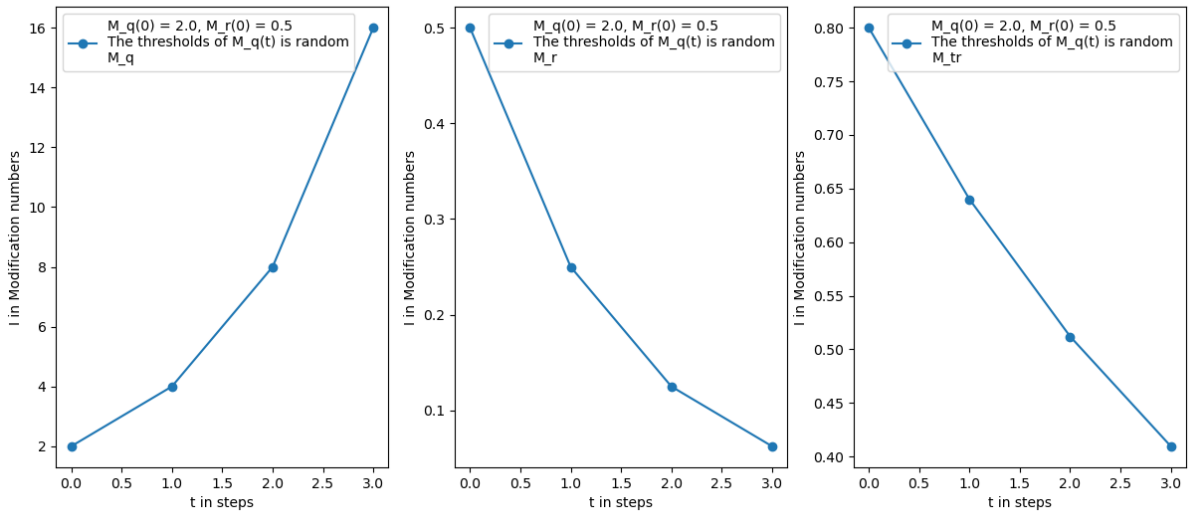
(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-16　初期値 M_q(0) = 2.0 と M_r(t) = 0.5 で M_q(t)のしきい値をランダムにした場合



(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-17　初期値 M_q(0) = 2.0 と M_r(t) = 0.5 で M_q(t)のしきい値をランダムにした場合

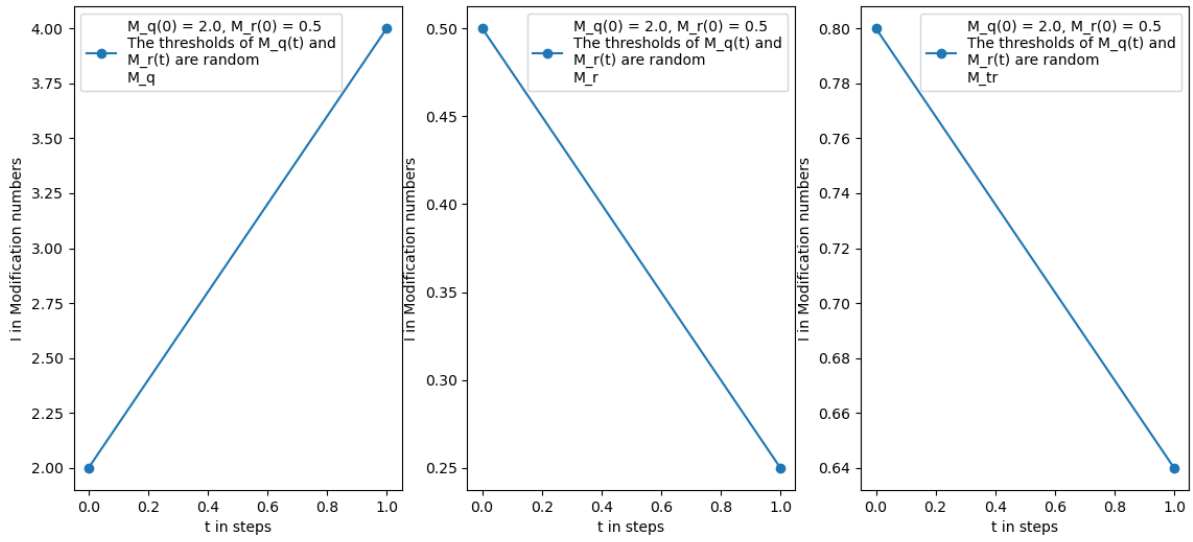


(a) q(t) (b) r(t) (c) tr(t)

図 3-18 初期値 M_q(0) = 2.0 と M_r(t) = 0.5 で M_q(t)のしきい値をランダムにした場合

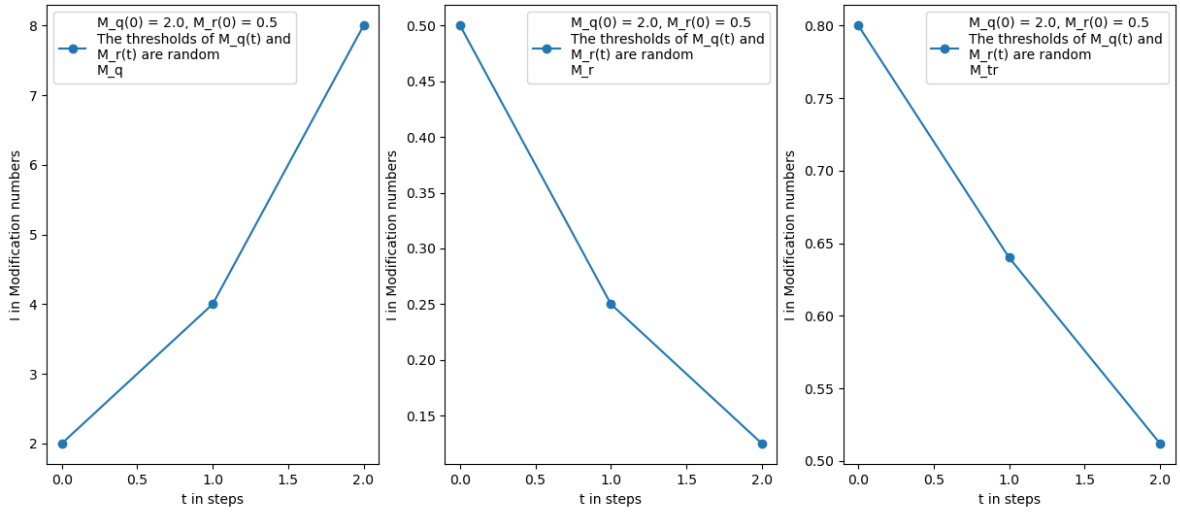
3.2.5 パーソナライズ（レベル3）

パーソナライズのレベル３では、レベル１とレベル２の両者のランダム化を行う。図 3-19 と図 3-20 に、各ノードに M_q（t）とそのコンテンツの信憑性および人間関係の信頼性のしきい値の両方をランダムに変動させる場合の結果を示す。両方の結果の一部は、他のノードへの送信と伝搬に対して、これまで以上により抑制されていることが示す。



(a) q(t) (b) r(t) (c) tr(t)

図 3-19 初期値 M_q(0) = 2.0 と M_r(0) = 0.5 で M_q(t) と M_r(t)のしきい値がランダム変数の場合



(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-20 初期値 M_q(0) = 2.0 と M_r(0) = 0.5 で M_q(t) と M_r(t)のしきい値がランダム変数の場合

これらの結果から、SNS の視点でみると、各ノードが SNS 内の他のノードへのメッセージのゲートキーパー(門番人)として機能する場合があることを意味している。ゲートキーパーとはまさに門番人として適切でないコンテンツを抑止し、信頼できない相手にメッセージを送らないことである。

3.2.6 パーソナライズ（レベル 4）

レベル 4 では、メッセージに対する肯定・否定の意見をレベル 3 のパーソナライズの要件に追加する。

メッセージに対する肯定あるいは否定を考える場合幾つかの疑問が沸いてくる。まず、肯定や否定が投稿者や発信者のモチベーションにどう関わるのかということである。

メッセージの肯定・否定をそのコンテンツだけで判定するのは非常に難しいことは、新垣ら[38]の研究が示している。多くの研究が判定にはコンテクスト（context, 文脈）とメッセージの送受信に関係する人間関係に影響を受けるとしている。今回導入したモデルでは、コンテクストに関わるのは、メッセージの内容、コンテンツを編集する処理であり、M_q(t)に集約されると考えられる。

「サッカーチーム X が 5 対 3 で勝利！」というメッセージを会員 a が受けた例で肯定・否定について考えてみる。

会員 a がこのメッセージを「肯定的」にとると、

- チーム X のファンである会員 b に知らせたいというモチベーションで「これで、X の優勝戦で競り勝とう！」などの応援のメッセージを付け加えて編集し投稿を準備する。
- 試合の内容で興味を惹かせたいというモチベーションであれば「後半戦までは同点で白熱。最後に u 選手のゴールで風が変わった。」のような試合のあらましを付け加えて編集し投稿を準備する。

何れも M_q(t)がプラスで増加するような傾向になる。

一方、会員 a がこのメッセージを「否定的」にとると、

- チーム X のファンである会員 b に知らせたいが、会員 a はライバルのチーム Y のファンであることから、「勝利はチーム Y ではわからないぞ！」などの対抗のメッセージを付け加えて編集し投稿を準備する。
- 試合の内容を否定したいモチベーションであれば「後半戦までは同点で互角。相手チームの v 選手の失策がなかったら勝てたかどうか」のような試合のあらましを付け加えて否定的な編集し投稿を準備する。
- チーム X のファンである会員 b に試合の結果まで知らせる必要はないというモチベーションで、会員 a はメッセージの投稿を取りやめる。

といったメッセージが考えられる。このとき、M_q(t)の極性は負になり、メッセージを投稿しない時は、M_q(t) = 0 となる。

以上を考慮して、レベル３に加え、M_q(t) = 0、正負の変動をモデルに適用して計算を進める。

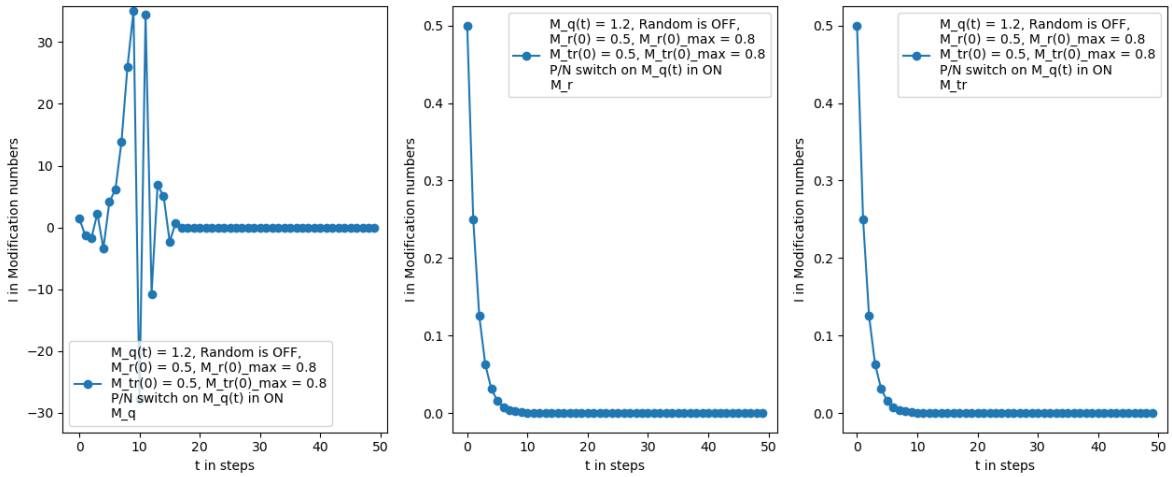
図 3-21 から図 3-25 は、各ノードが受信した情報と固定された M_q（t）およびしきい値に対して肯定的または否定的な意見を持っている結果の一部を示す。

正または負（P / N）の意見は、M_q（t）の極性によって与える。また、M_q（t）の値は、標準正規分布で $-1.0 < M_q\,(t) < 1.0$から選択する。この状況には M_q（t）= 0 が含まれており、他の会員へのメッセージングを停止し、メッセージを投稿しないことを意味する。

計算の結果をみると正負（P / N）の意見による M_q（t）の動作は、図 3-21 から図 3-25 で減衰振動や拡散の減少が見られる。

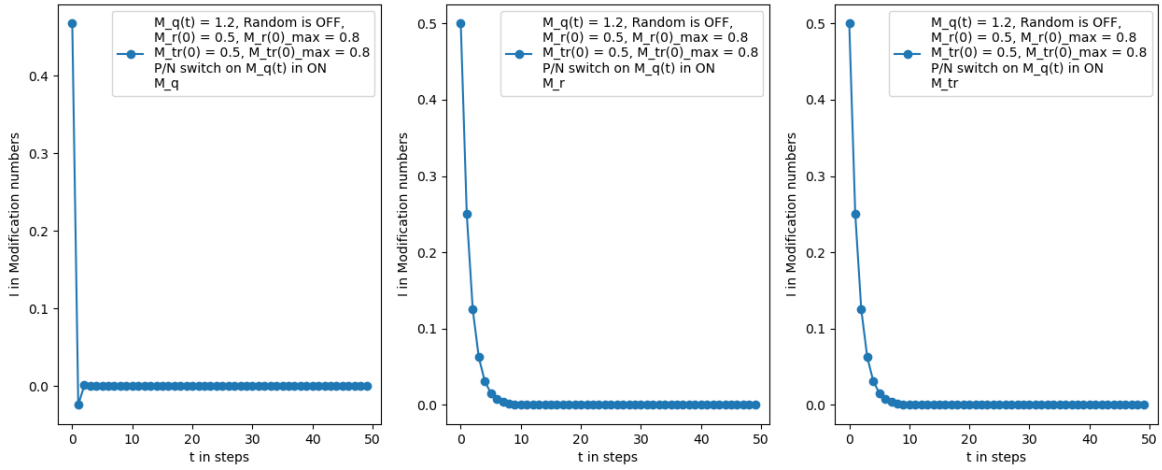
M_q（t）の極性が同じである場合、拡散は進行するが、それ以外の場合は、急速に収束することが観測できる。

図 3-21 から図 3-25 の計算結果は、正負（P / N）の肯定・否定の意見を持ち、ランダムな M_q（t）、および M_q（t）、M_r（t）、M_tr（t）のランダムなしきい値を持つ人間のメッセージングに近いことを示している。各ノードで同時にランダムなモチベーション変数を活性化するため、図 3-21 と図 3-23 の結果を見つけることは稀で、ほとんどの結果は急速に収束し、多くの計算で拡散は発生しないことが分かった。



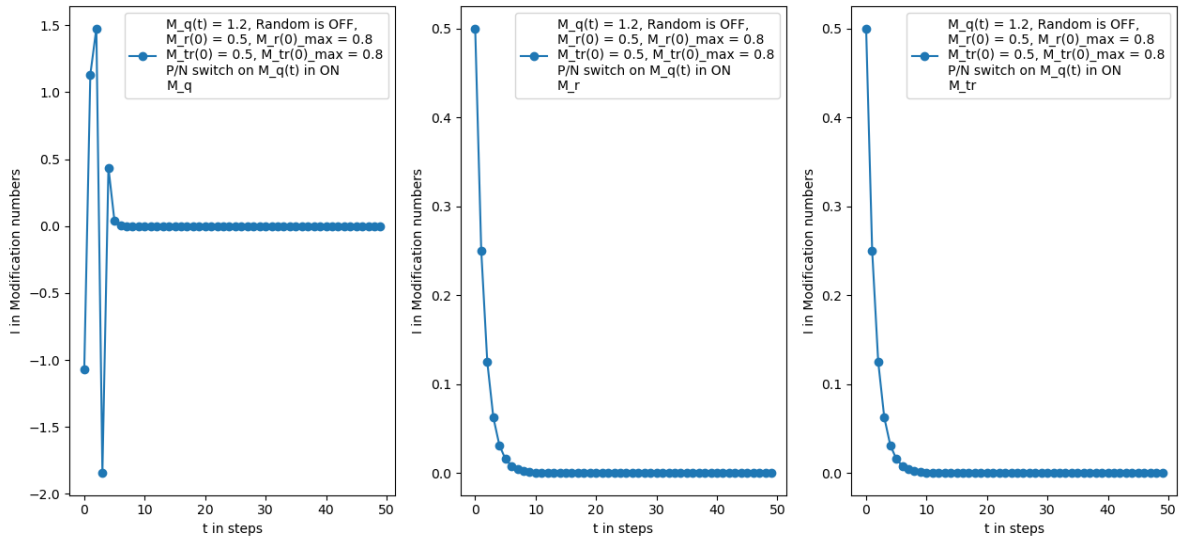
(a) q(t) (b) r(t) (c) tr(t)

図 3-21 M_q(t) = 1.2 でランダムスイッチは OFF、初期値 M_r(0) = 0.5, M_r(0)_max = 0.8, M_tr(0) = 0.5, M_tr(0)_max = 0.8 として、 P/N スイッチを ON にした時の結果の一部



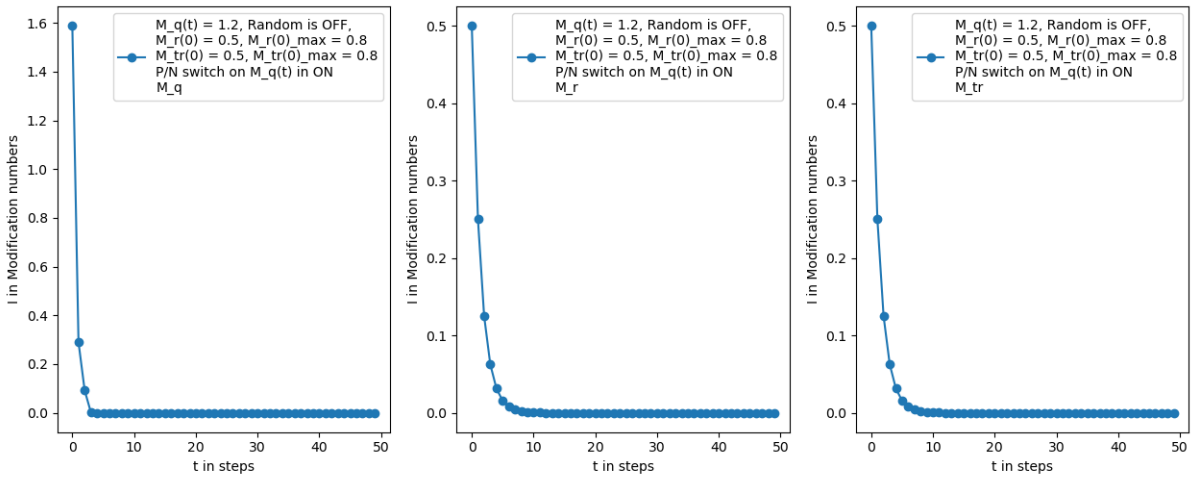
(a) q(t) (b) r(t) (c) tr(t)

図 3-22 M_q(t) = 1.2 でランダムスイッチは OFF、初期値 M_r(0) = 0.5, M_r(0)_max = 0.8, M_tr(0) = 0.5, M_tr(0)_max = 0.8 として、 P/N スイッチを ON にした時の結果の一部



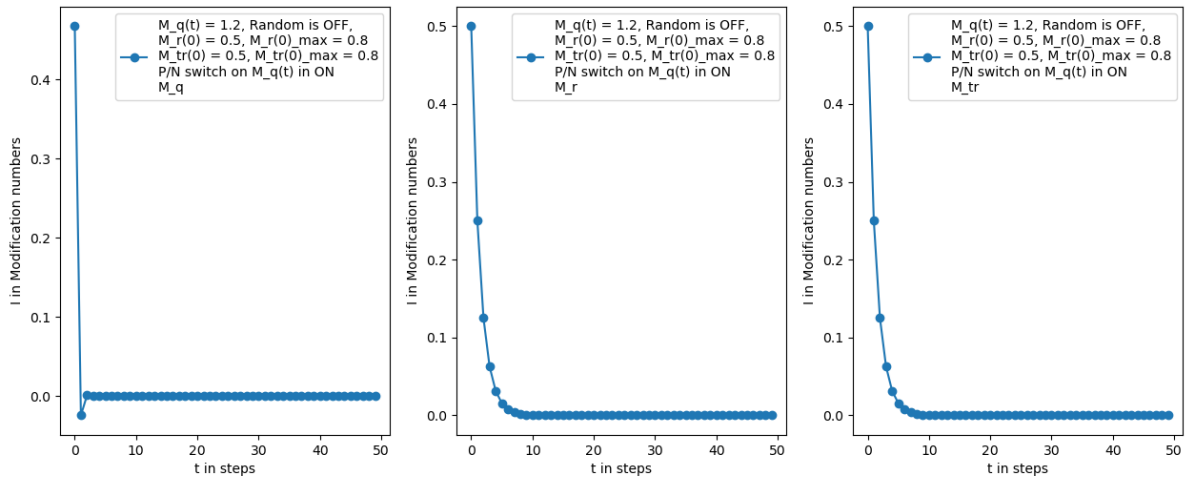
(a) q(t) (b) r(t) (c) tr(t)

図 3-23 M_q(t) = 1.2 でランダムスイッチは OFF、初期値 M_r(0) = 0.5, M_r(0)_max = 0.8, M_tr(0) = 0.5, M_tr(0)_max = 0.8 として、 P/N スイッチを ON にした時の結果の一部



(a) q(t) (b) r(t) (c) tr(t)

図 3-24 M_q(t) = 1.2 でランダムスイッチは OFF、初期値 M_r(0) = 0.5, M_r(0)_max = 0.8, M_tr(0) = 0.5, M_tr(0)_max = 0.8 として、 P/N スイッチを ON にした時の結果の一部



(a) q(t)　　(b) r(t)　　(c) tr(t)

図 3-25 M_q(t) = 1.2 でランダムスイッチは OFF、初期値 M_r(0) = 0.5, M_r(0)_max = 0.8, M_tr(0) = 0.5, M_tr(0)_max = 0.8 として、 P/N スイッチを ON にした時の結果の一部

3.2.7 パーソナライズ（レベル 5）

パーソナライズのレベル４までは、各ノードは自ら投稿したメッセージを記憶していない。また、投稿を見た受信者についても知り得ない。しかし、SNS のシステム構成では、投稿したメッセージをフォローアップ（追従）したり、自分の投稿をタイムラインに残す機能があるので、自分の投稿したメッセージを見たり、フォローする会員を知り得ることができる。

そこで、レベル 5 では、投稿という経験を記憶し、モチベーションのしきい値を記憶に応じて変動させることを試みた。

投稿とその記憶（履歴）とはどのような関連があるのか。

「サッカーチーム X が５対３で勝利！」というメッセージを会員 a が受けた例で記憶について考えてみる。

会員 a がこのメッセージを見た段階で、

- 自分がこれまでサッカーチーム X についてどのようなメッセージを投稿したのか記憶あるいはタイムラインを探る
- 「サッカーチーム X は今年も優勝か？」といった投稿を行っていたことがわかる。
- 過去の投稿で会員 b がこれに対して肯定的なメッセージ「チームワークのよさが今年はセットプレーの成功で分かり優勝は狙える」と投稿していた。
- 会員 a は自分の過去のメッセージと会員 b がこのメッセージに肯定的でメッセージを投稿していたことがわかったので、モチベーションを上げて「サッカーチーム X が５対３で勝利！」に肯定的なメッセージを投稿することにした。

といった、過去の情報（自分の過去の投稿メッセージやフォローした会員 b、さらにそのメッセージ）から肯定的なメッセージの投稿をする気になった。このように、記憶は、投稿するモチベーションに影響を与え、メッセージをレベル４までのパーソナライズも含んで、より現実の SNS の動態に近づくことになる。

図 3-26 は、 t = tn で一旦メッセージの拡散が停止した状態を示している。次のイベント E2 が起こった場合、ノード(a)は、過去 t = 1 でメッセージを投稿し、ノー

ド(a)とリンクがあるノード(b)がこれにコメントをしている。ノード(a)は t = 1 のメッセージを記憶していたことから、イベント E2 に対して、メッセージを投稿する準備を行う。

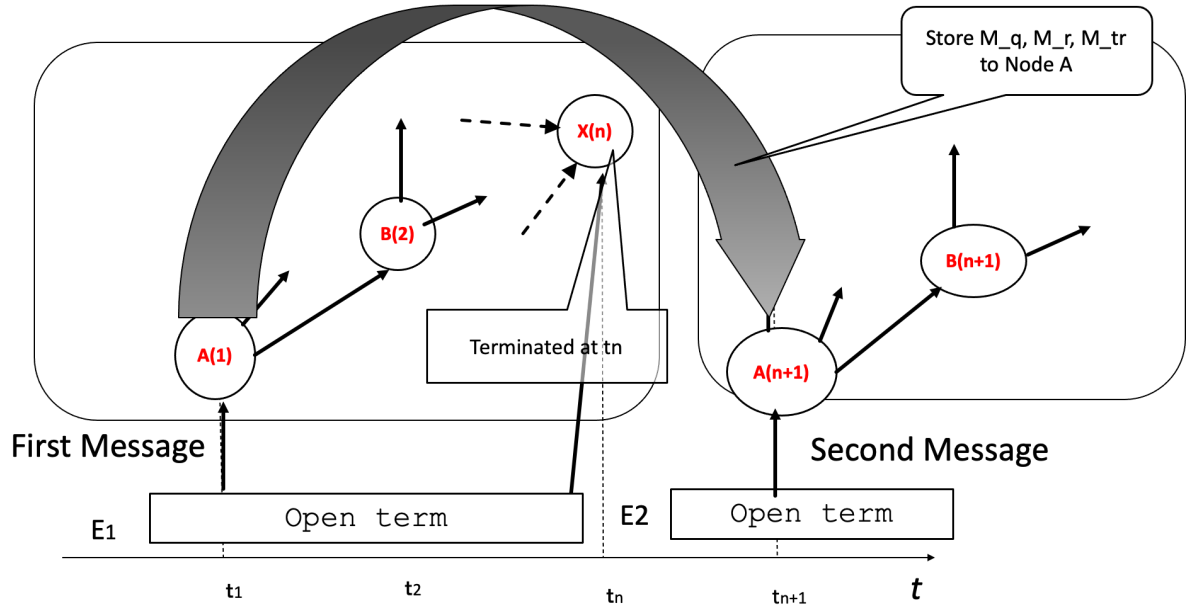


図 3-26 イベントのコンテンツの体験を記憶し、2 番目以降のイベントのしきい値を形成

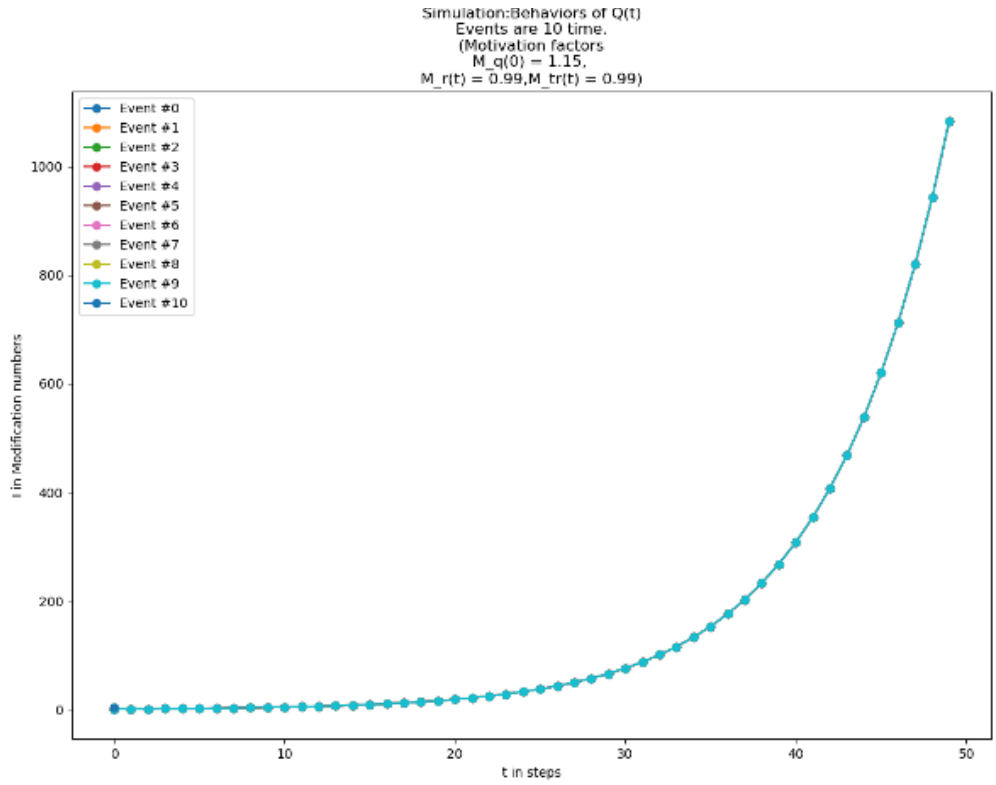


図 3-27 参照ケースとしての M_q (0) = 1.15 (M_r (t) および M_tr (t) は定数)。 イベント数は 10 回

さて、記憶は、モデルの M_q（t）、M_r（t）、M_tr（t）のしきい値に影響を与えるため、比較のために参考となる状態をまず計算することにした。参考のデータは M_q（0）= 1.15 として、メッセージを拡散させ、M_r（t）と M_tr（t）はしきい値の変更をしないように定数とした。 イベントは 10 回おこなった。

結果は図 3-27 のように指数関数で拡散し、しきい値は変化がないため、10 回のイベントが全て重なっている。

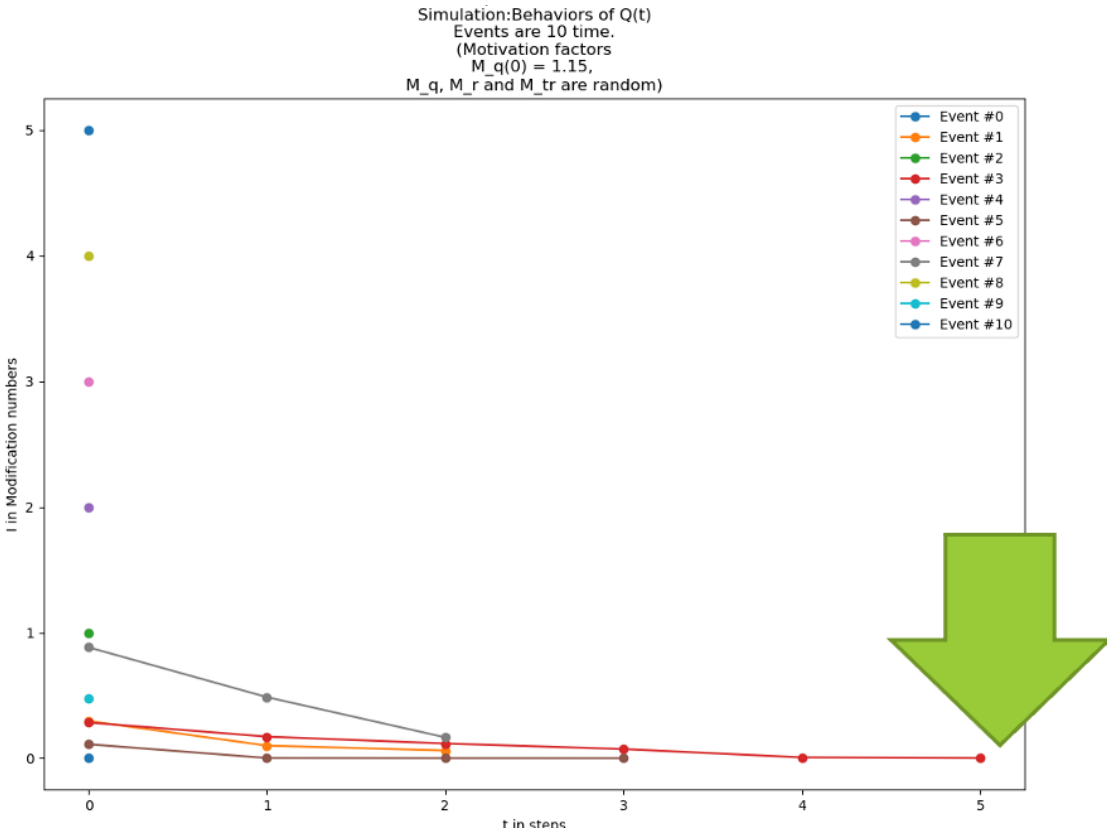


図 3-28　図 3-27 と同様に M_q（0）= 1.15。M_r（t）と M_tr（t）はランダムで、イベント＃1 から記憶していく場合

図 3-28 は、M_q（0）= 1.15 として拡散させ、M_r（t）と M_tr（t）はランダム化するイベント＃1 から順次記憶していく場合で明らかに、拡散が低下している。さらに、図 3-29 から図 3-31 まで、M_q（t）のしきい値を直近の記憶に対して 30%、50%と 80%に上昇させた場合の結果を示す。

- 計算の結果はレベル 4 のしきい値のランダム化の結果と同様に、メッセージの拡散は少ない。
- 履歴のあるノードでは、2 番目以降のイベントでメッセージングを拡散することが困難になっている。
- しきい値を 30%、50%、80%と上げる（嵩上げする）と、メッセージが拡散する傾向がでるが、非線形増加である。

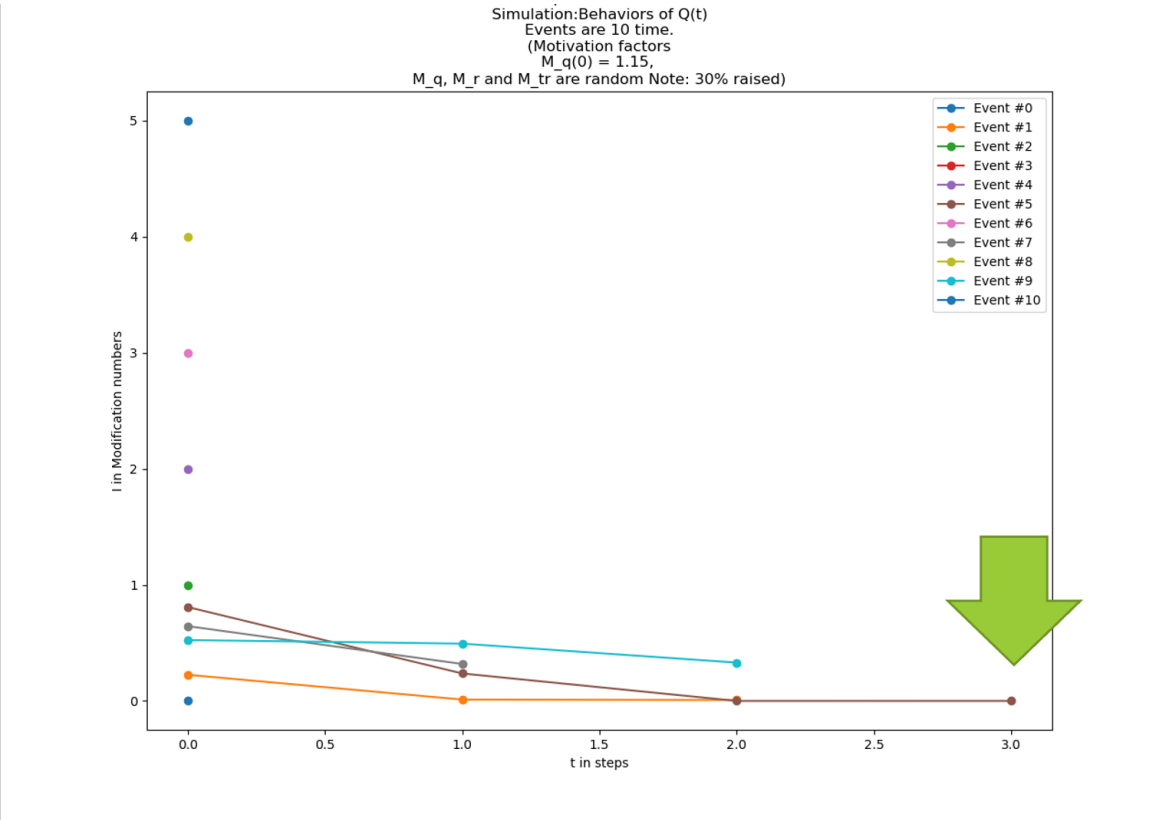


図 3-29 M_q (0) = 1.15、M_r (t) およびM_tr (t) はランダムであり、イベント#1のメモリがあり、M_q (t) のしきい値が30%高く嵩上げ

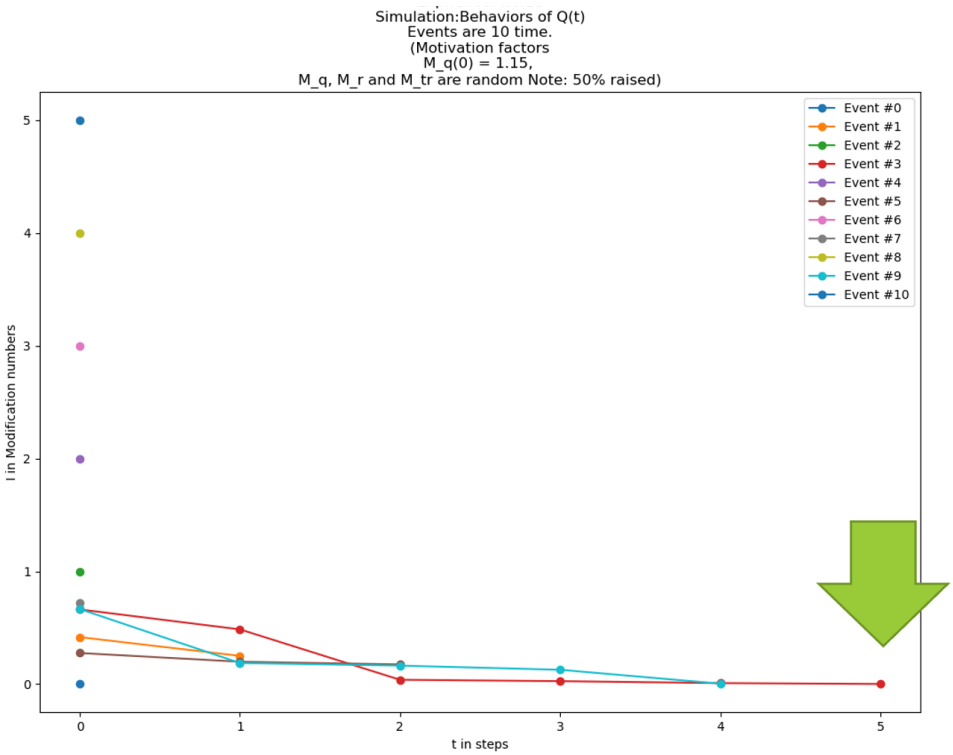


図 3-30 M_q (0) = 1.15、M_r (t) およびM_tr (t) はランダムであり、イベント#1のメモリがあり、M_q (t) のしきい値が50%高く嵩上げ

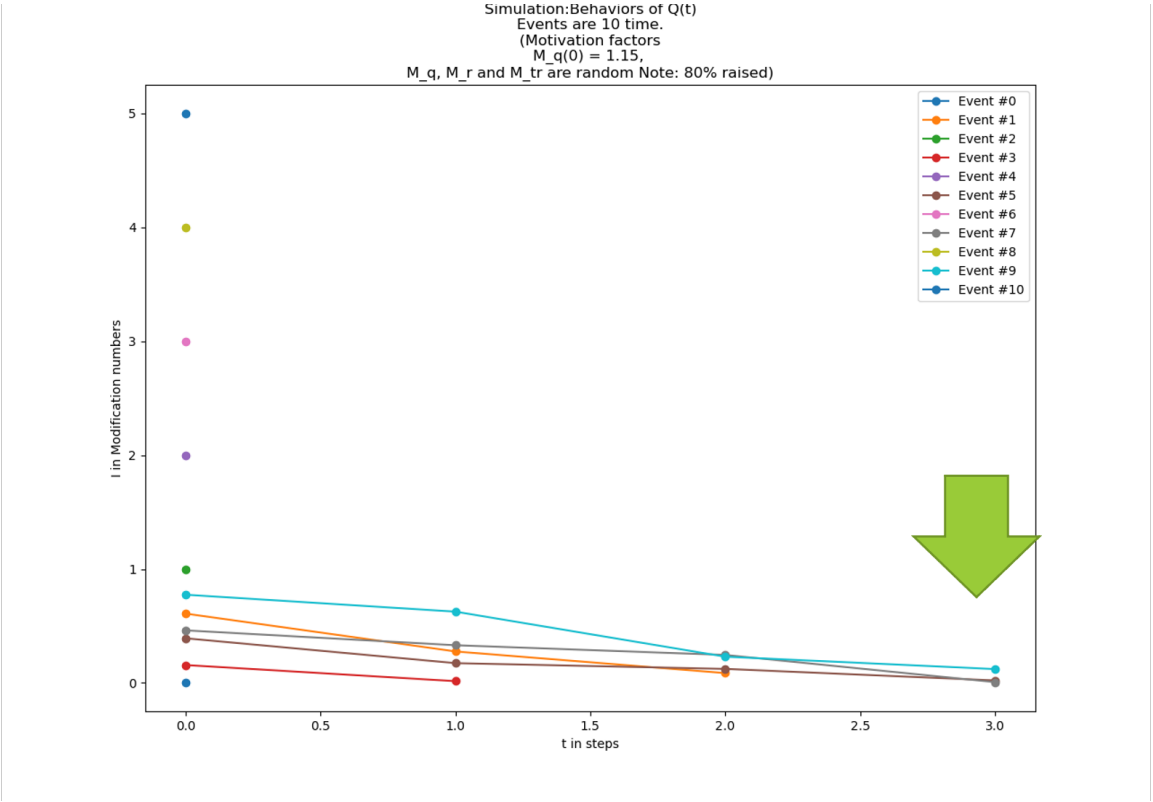


図 3-31 M_q (0) = 1.15、M_r (t) および M_tr (t) はランダムであり、イベント#1 のメモリがあり、M_q (t) のしきい値が 80%高く嵩上げ。

記憶したしきい値の嵩上げで拡散を増加させることは、記憶を繰り返し呼び起こす恣意的な広告宣伝など似ている。繰り返しの広告手法は、嵩上げ現象に応用できるかもしれない。

これらの結果を見ると、しきい値をランダムにする効果と同様、履歴によるメッセージングの影響は意外に大きい。言い換えれば、メッセージを拡散する場合にコンテンツの信憑性、人間関係の信頼度を過去の履歴も参考にすることで、不用意な拡散を抑制する効果があることが分かる。

社会現象として buzz（話題になること）を過剰に起こさせないためにも、コンテンツの検証、信頼関係のある人間関係の確証、さらに過去の投稿の検証が重要であることがこれらの結果から分かる。[9]

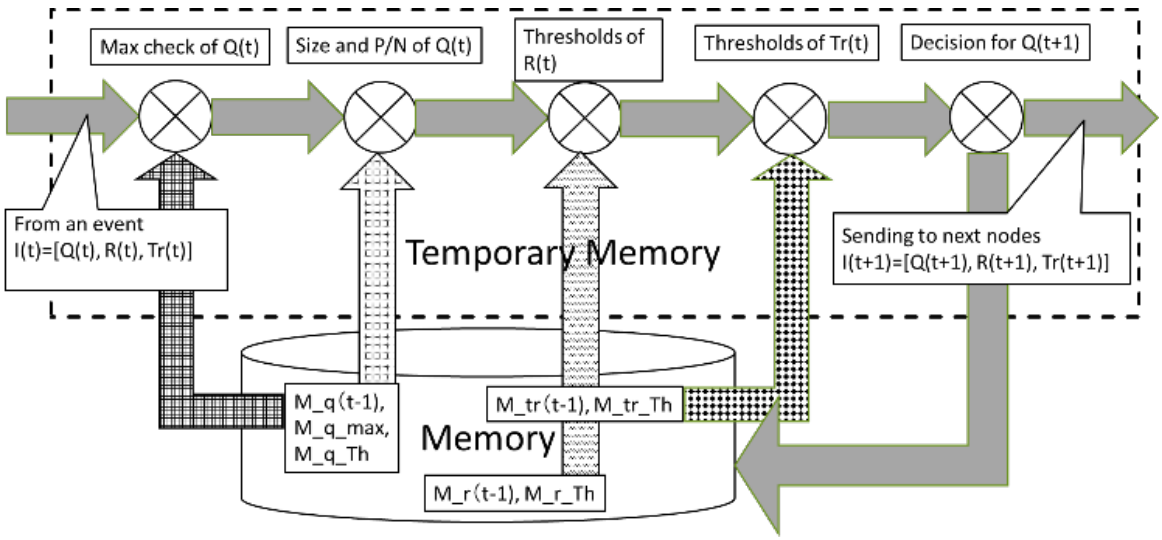


図 3-32　レベル 1 からレベル 5 へのパーソナライズした場合

導入したモデルのノードをパーソナライズすることで、段階的に SNS で行っている現象を再現してきた。図 3-32 は本モデルの 1 ノードに注目し、レベル 1 からレベル 5 へパーソナライズした場合、一連の判断を要約した。

図 3-32 は 1 つのノードを示している。左からの入力に対して

- 過去の Q(t)の最大値との比較判定
- 過去の Q(t)の大きさと P/N と比較判定
- 過去の R (t)のしきい値との比較判定
- 過去の Tr(t)のしきい値との比較判定
- 設定した Q(t)の投稿と投稿したメッセージの記憶

といった各段階を通過して表現される。それぞれの判断はノードのモチベーションに依存する。

第４章　個人から集団へのダイナミクス

第３章では、１つのノードあるいはイベントからメッセージが拡散するイベント・ドリブン・モデルで、従来のインターネットの通信機能と SNS で扱うヒューマン・コミュニケーションで分けて考察してきた。また、ノードやイベントの情報源を中心にメッセージの拡散に注目してきた。

本章では、一人の会員からイベントの情報を拡散するのではなく、会員が同時に他の会員やイベントのメッセージを受けた場合、メッセージを会員で共有しながら展開する現象をモデル化することを考える。

４.１　同一意見の集団の形成

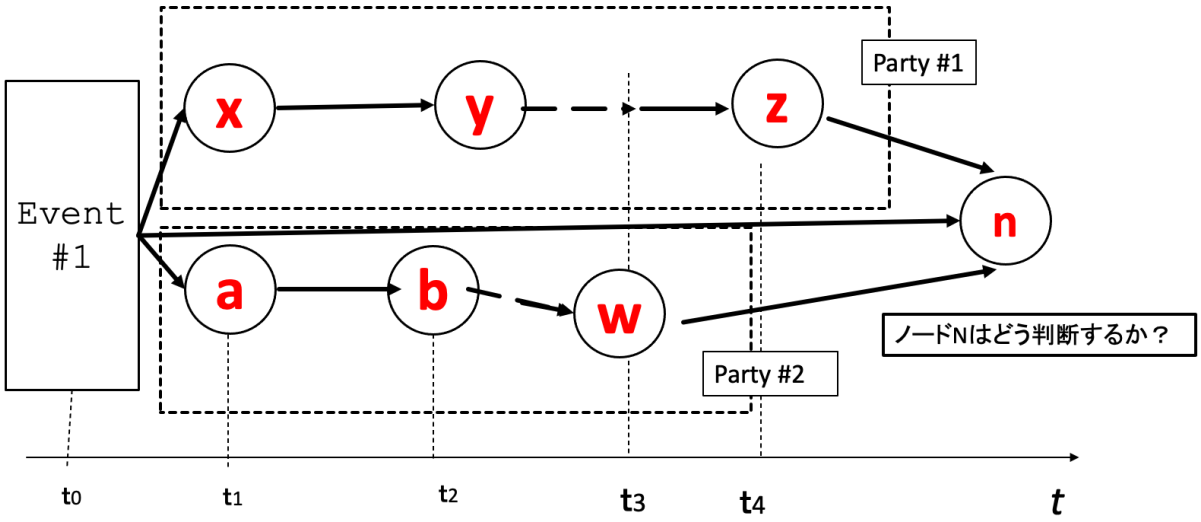


図　4-1　２つの集団の形成と第３のノード(n)の立場

３.１.３では、単一情報源（イベントやノード）を扱い、受信者は複数いるが、最初にメッセージをみるのは単一のノード(a)だけで、他のノードは時間遅れでイベントのメッセージをみるか、ノード(a)のメッセージをみるかであった。今回考えるモデルは、この時間遅れがない状態で、ノード(a)とノード(x)が同時にイベントのメッセージを受信した場合を考える。SNS では厳密に同時での受信はないが、ほぼ同一時刻に受信者がメッセージを見ることは起こる。

図 4-1 はこの状況を示したもので、ノード(a)はリンクのあるノード(b)にイベントの情報を編集してメッセージとして投稿する。このリンクの連鎖は、ノード(w)が t = t3 でメッセージ受け取るまで続くとする。ノード(w)はメッセージを投稿して、リンクのあるノード(n)が受信できる状態となる。

一方、ノード(x)は、ノード(a)と同様にリンクのあるノード（y）にイベントの情報を編集してメッセージとして投稿する。このリンクの連鎖は、ノード(z)が t = t4 でメッセージ受け取るまで続くとする。ノード(z)はメッセージを投稿して、リンクのあるノード(n)が受信できる状態となる。

さて、ノード(a)からノード(w)まではメッセージをリンクのあるノード間でつなぎ、同一意見の集団（party）を形成することになる。同様にノード(x)からノード(z)まではメッセージをリンクのあるノード間でつなぎ、同一意見の集団（party）を形

成することになる。ただ、ノード(n)から見て、この両集団意見が違っていると、どちらの集団に入るかといった疑問が生まれる。もちろん、どちらの集団にも入らない場合もある。そこでノード（n）の動態を扱うことを考える。

３.１.３では SNS で会員のメッセージは会員の個人的なモチベーションによって生成されてきた。会員の履歴を使う場合も他のノードをどう過去に扱ったかという個人的な意見やモチベーションだけでメッセージングした。実際、初めて SNS に加入した会員は暫く、他の会員がコンテンツによってどのような集団に扱われ、所属しているのかについて無頓着であり意識が少ない。投稿に慣れ、他の会員との交流が進むと、会員同士が交流によってできる仲間という小集団ができる。小集団では、お互いのメッセージに対する意見が予想でき、コンテンツの信憑性や仲間の信頼性は上がっていく。図 ４-1 はこのような集団の 2 つが、現時点ではノード(n)ではどちらに所属するかは不明な状況であるとも言える。

ここで重要なことは、集団に属さない間は、ノードはそれぞれ異なったメッセージを投稿するが、集団に属する会員からのリンクが生まれた時、メッセージを自己のモチベーションでの生成だけでなく、集団のメッセージの影響を受けて、編集加工されることである。このことは、これまではメッセージをノード内の処理でモチベーションに応じて扱ってきたというミクロ的な視点であったところから、集団的な視点でノードの外部からの意向（集団的意識）に沿うようにメッセージを処理するマクロ的な視点が入ることになる。[7]

４.２　能動的ノードの利用

３.１.３でのモデルでは、各ノードでは自己判断は初期設定で一括して行われるだけの受動的ノードである。これでは、他のノードからの影響をダイナミックに記述できない。各ノードがメッセージを受け取った後も独立してメッセージの処理を行える能動的ノードに変える必要がでてきた。

そこで、より現実のヒューマン・ネットワークに近づけるために、受動的なノードからノード自体が判断してコンテンツを編集する能動的ノードを導入することにした。

４.３　集団のメッセージ空間

会員のメッセージによってできる集団とはどのようなものであろうか。

SNS では多くの誹謗中傷の事件のように、１つのメッセージを意図的に（モチベーションを持って）拡散する場合、会員は SNS で繋がっている友人や知人を使ってメッセージを拡散する。この SNS で繋がっている友人や知人は局所的な集団と見なすことができる。このとき発信者の会員によるルールや規則でメッセージは拡散される。このような会員の集団からのメッセージの拡散によってメッセージを受けた受信者から次々と集団が形成されている。さらに集団の会員が別の集団と繋がることから集団が生まれる。

前章でノードにモチベーションによるメッセージングの判断を行えるような情報の伝搬を考えてきた。これは、SNS でいえば会員の家族、友人や知人との交流を示し、局所的な個人の領域でコミュニケーションを扱うことに相当する。

実際には SNS で会員の嗜好や感情、信頼性、思想や価値観に応じて交流を頻繁に行う集団が形成されることが観測される。その背景には認知科学での欲求とモチベ

ーションの関係で説明されているが、集団の帰属性や他の集団との関係によって集団内部の会員のメッセージングにも影響を与える。[17]
次に集団の動態を計算で観測してみる。

４.４　集団の形成と拡散モデル

これまでのヒューマン・コミュニケーションのメッセージ空間を利用して、その表現で能動的ノードをもつエージェントモデルで行うこととした。エージェントモデルの利用は、各ノードをエージェントとして扱うことで能動的ノードを表現でき、かつ、他のノードからの影響も表現できるからである。
今回導入するエージェントモデルは、３.１.３のようなノードが静止してメッセージが拡散していく静的なモデルとは異なり、多くのノードが 2 次元平面上をランダムウォークしている動的なモデルである。

４.４.１　導入するメッセージ空間

ヒューマン・コミュニケーションでのメッセージ空間を 2 次元平面での 2 つのノード間の距離と帰属する集団に対する状態で示すことにする。
ノード間の距離は、会員同士の親密度に相当し、距離が短いと親密であるとする。さらに、SNS では受信者の有無にかかわらずメッセージを発信し、受信側は自らのモチベーションでこれを受信して反応する。この反応は、ここでのモデルでは発信者はランダムウォークすることで、受信者とはランダムに遭遇することで再現する。
３.１.４で導入したイベント・ドリブン・モデルでは、すでにリンクを設定したノード間でのメッセージングを考えたが、これでは、リンクができる説明ができない。そこで、ノードとノードの距離を使って、近いほど会員同士の親密度が高いとし、リンクが容易に形成できるとした。
発信者は、メッセージの信憑性（reliability）と発信者との人間関係での信頼性（credibility）でメッセージをさらに送信するか、編集して送信するか、送信しないといった判断を行う。さらに、各ノードは図 ４-2 に示すように、メッセージの信憑性（reliability）と発信者との人間関係での信頼性（credibility）の積(6)式を信頼度レベル（Trust level）としてメッセージの到達距離に比例させた。つまり、信頼度レベルが高いとメッセージも遠方にまで到達するとした。

$$Trust\ level = Reliability * Creditability \tag{6}$$

４.４.２　影響度 R の範囲

信頼度レベル（Trust level）は 2 次元平面の変換で単位時間あたりの移動距離で速度に相当する。
一方、影響度 R の範囲は、各ノードがもつ他のノードへ影響を与える範囲である。影響を受けることはノード間にリンクが確立したことで、メッセージが届き、受信したノードが発信者のメッセージが検知できる範囲である。同図では半径 R の円内の範囲である。なお、影響度 R の範囲とリンクの関係を説明するためにノード(A)の影響範囲は省略している。
ノード（B）がノード（A）に信頼度レベルの速度で接近した場合、ノード（A）はノード（B）の R の内部にはないので、影響を受けず、そのままランダムウォーク

を続ける。ノード（B）が移動して、図 4-3 のようにノード（A）が影響度 R の範囲の内部にあるとき、ノード（A）とノード（B）はリンクを確立し、メッセージの交換ができるとする。

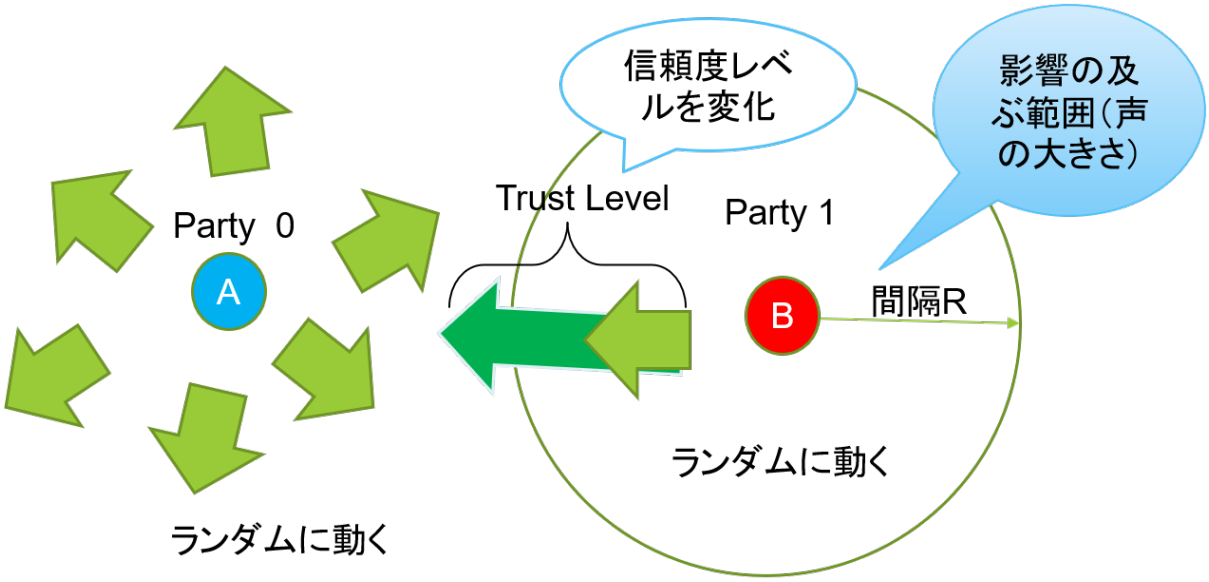


図 4-2 導入したメッセージ空間、影響度 R の範囲とリンクの関係を説明するためにノード(A)の影響範囲は省略している。

実際の SNS では、A さん（ノード（A））がメッセージを投稿後、B さん（ノード（B））に A さんが自分の投稿（メッセージ）を閲覧することを許せば、リンクが確立していることになる。B さんの信頼度レベルは、ノードが 2 次元平面で移動する速度であることから、速度が上がることに比例して接触するノード数が増える。これは、SNS で B さんの信頼性が高いほど、そのコンテンツが多くのノードに注目されることに相当する。

影響度 R の範囲は、他のノードとのリンクを確立する範囲であり、同時にリンクを通してメッセージを共有できる距離（範囲）を示している。この信頼度レベル、影響度の範囲とリンクの関係は、SNS で言えば、信頼度レベルが、メッセージで影響を受ける会員数に比例しており、リンクの数にも比例する。いわば、信頼度レベルは、群衆の中での「声の大きさ」で、声が聞こえる「事実」がリンクに相当する。影響度 R の範囲は「声の届く範囲」と例えることができる。群衆の中の声と同じで、いくら大きな声であっても、声の届く範囲にいなければ声は聞こえないのである。

4.4.3　属する集団によるメッセージへの影響

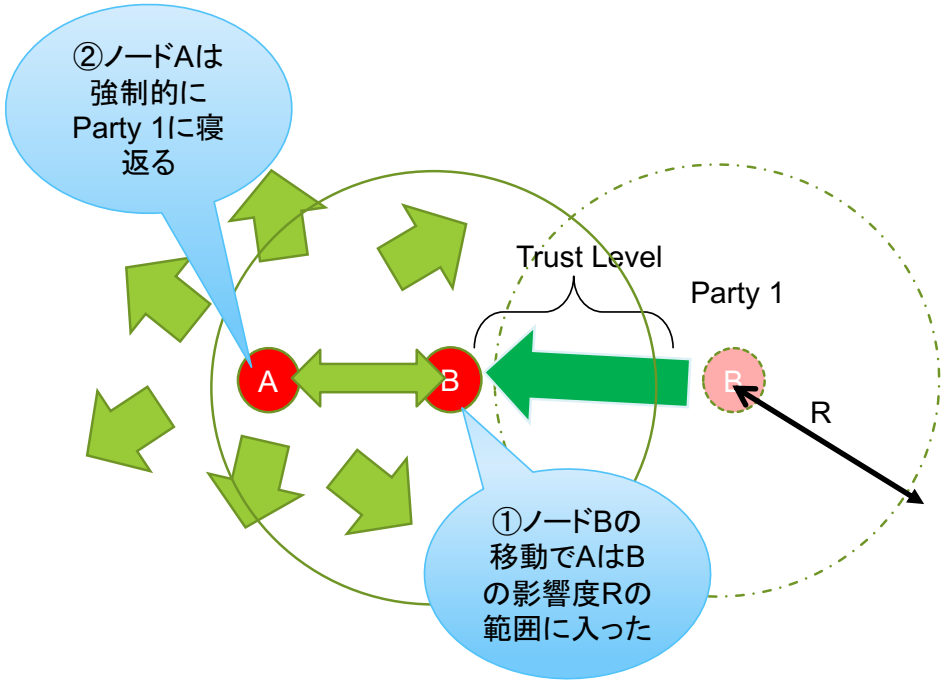


図　4-3　影響度 R の範囲とリンクの確立

図 **4-3** でノード（A）に近づいたノード（B）を考えよう。

① ノード（B）の移動でノード（A）は B の影響度 R の範囲に入った。このときの移動速度は信頼度レベルに比例している。

② ノード（A）は強制的にノード（B）の属する集団（パーティー（1））に「同調する戦略」をとるならば、パーティー（0）からパーティー（1）へ寝返ると仮定する。

以上の内容をメッセージングで考えると図 **4-3** でノード（A）に近づいたノード（B）の影響は、SNS ではノード（B）のメッセージをノード（A）が閲覧することに相当する。これまでの 3.1.3 のモデルであれば、メッセージの信憑性と B との人間関係の信頼度でノード（A）が投稿しようとするコンテンツが決まる。しかし、ノード（A）はパーティー（0）に属し、ノード（B）は別のパーティー（1）に属しているために、決まったコンテンツをそのまま投稿せずに何らかの編集が行われる。

例えば、ノード（B）のメッセージをそのまま送信する（同調する）場合や反対のメッセージを送信する（否定する）場合が発生する。このように、従属する集団というマクロ的な要因でメッセージを送信するモチベーションが変化する。

4.4.4　影響度 R の範囲のメッセージへの影響

4.4.3 で帰属する集団のメッセージへの影響を述べたが、他にもメッセージに影響を与えるものが、影響度 R の範囲である。

影響度 R の範囲の導入は、他のノードへの影響を与える範囲であることから、受信者が次に投稿しようとするメッセージに影響を与えることになる。

例えば、他のノードの影響を受けないで帰属する集団に同調するメッセージを投稿しようとした時に、帰属する集団に反対する別の集団のノードが先にメッセージを投稿すると、これまでのメッセージを編集して、帰属集団にそった意見や反対集団への攻撃文に変わることもあるだろう。このように、影響度 R の範囲はノードから見えるメッセージ空間の視界であって、視界の外からは影響を受けない。

ここでは、メッセージによる交流でできた集団（メッセージ集団）が個々のメッセージングにどのような影響を与えるか、集団が複数あった場合はどうかをシミュレーションによって考察する。

4.5　シミュレーションと結果

エージェントモデルによるシミュレーションで各ノードは能動的ノードで構成し、メッセージの信憑性、人間関係の信頼性、帰属する集団の属性を持たせ、影響度 R の範囲を変えることで、個人から集団へのダイナミクスを考察する。

最初に今回導入したエージェントモデルの検証を自明なパラメータで行う。

4.5.1　シミュレーターの検証と集団戦略

シミュレーションの検証は 1 ノードだけのメッセージ集団に接触した場合を示す。図　4-4 は、信憑性、信頼性がともに 100%で 50 個の同じ意見のメッセージ集団で R=0.1 である。このとき 1 つのノードが接触した場合、R=0.1 の範囲内にあって自分の集団と異なるノードは自分と同じ集団に帰属させる（寝返り）するものとする。図 4-5 図　4-4 のように時間とともに最初の 1 ノードのメッセージが他のノードを寝返らせながら拡散していくことがわかる。この結果は、自明であり、このシミュレーションが検証されたことになる。

ここで、「自分の集団と異なるノードは自分と同じ集団に帰属させる（寝返り）する」ことは、集団で発生するマクロな戦略である。これについては 4.6 で論じるが、ここでは、接触した相手を同一意見に変え、自分の集団に組み込み吸収する戦略とする。

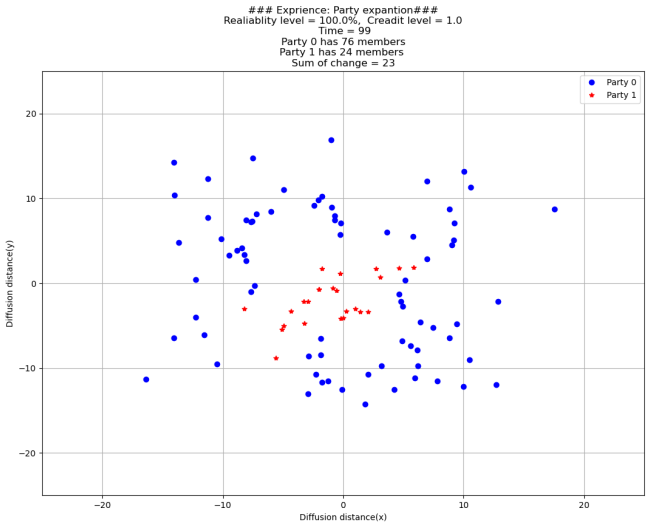


図　4-4 コンテンツの信憑性：100%、同じパーティーの信頼性：100%、N = 50 nodes　T = 100 steps、最初は Party1 が 1 node

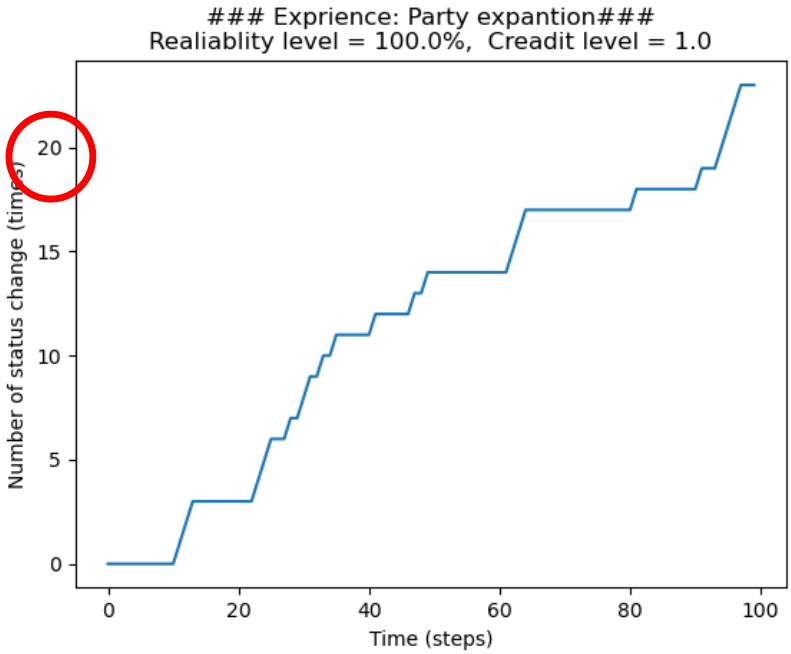


図 4-5 寝返り率の時間的変化

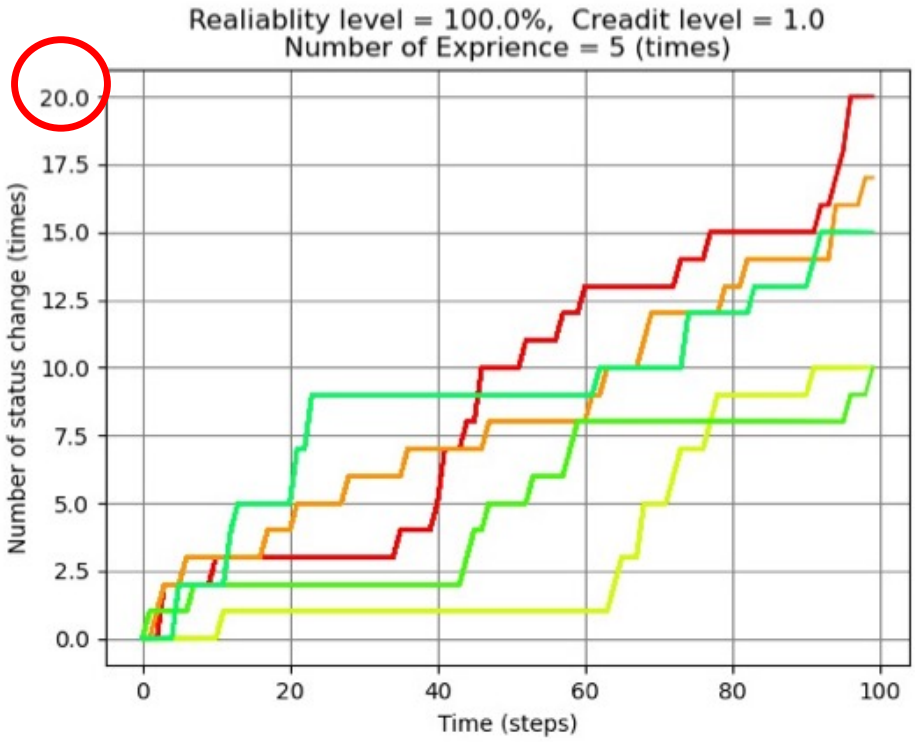


図 4-6 コンテンツの信憑性：100%、同じパーティーの信頼性：100%、N = 50 nodes、T = 100 steps 最初は Party 1 が 1 node である場合である。実験回数で色分けした。

なお、図 4-6 は、パーティー（1）のノードが相手を寝返らせたノード数を縦軸に取り、時間経過を見たものである。
図 4-5 は 1 回のシミュレーションの結果であるが、図 4-6 はさらに 4 回のシミュレーションの結果を示す。

4.5.2　信頼度レベルを変える

同じ影響度 R の範囲で、今度は、信頼度レベルを 50%に変えると、図 4-7 のように、寝返り数は増加するものの、図 4-6 と比較して、約 50%程度の寝返り数となる。信頼度レベルが低いと、同じ時間でも寝返り数も下がることから、寝返りの速度が低くなると言える。メッセージの内容や人間関係に信頼度が低いと、寝返りの速度が落ちることがわかる。信頼度レベルが低いと、ノードの移動が少なく、接触するノードも減るからである。

信頼度レベルを小さくする、即ち信憑性や信頼性が低下すると、図 4-10 のように時間当たりの寝返り数が低下する結果となった。なお、図 4-10 の設定でのノードの様子の一例を図 4-9 に示した。図 3-10

図 4-7 と図 4-8 において寝返り数の最大値を比較することで、信頼度レベルが低下すると、それ自身のメッセージを拡散することは困難であるという現象と見ることができる。

図 4-8 は、図 4-6 と同条件でシミュレーションを 15 回行ったが、傾向は同じであった。図 4-10 も、信頼性を 90%としたが、傾向は同じである。

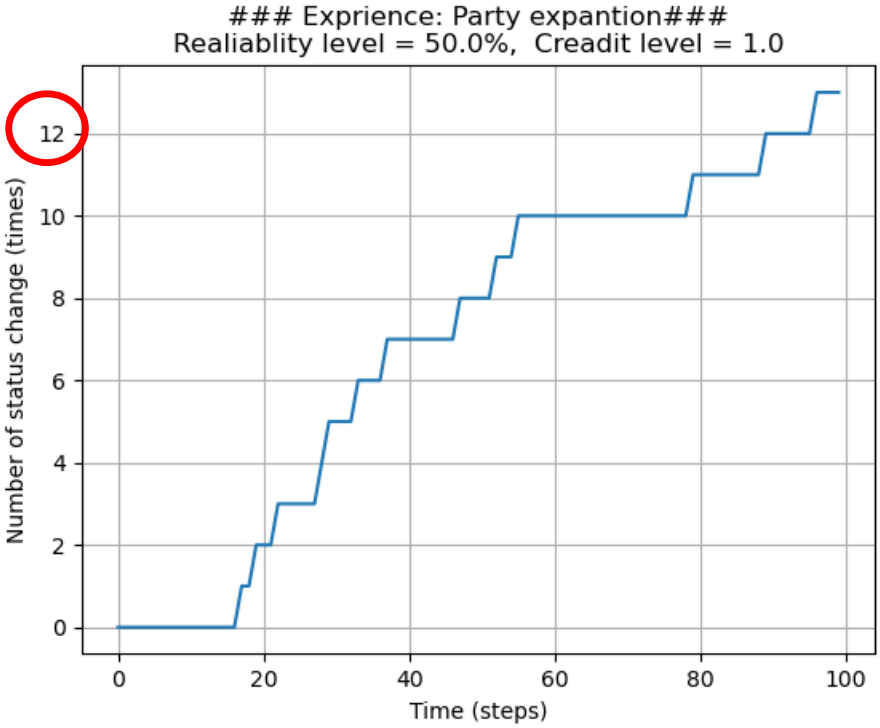


図 4-7 **コンテンツの信憑性：50%、同じパーティーの信頼性：100%**

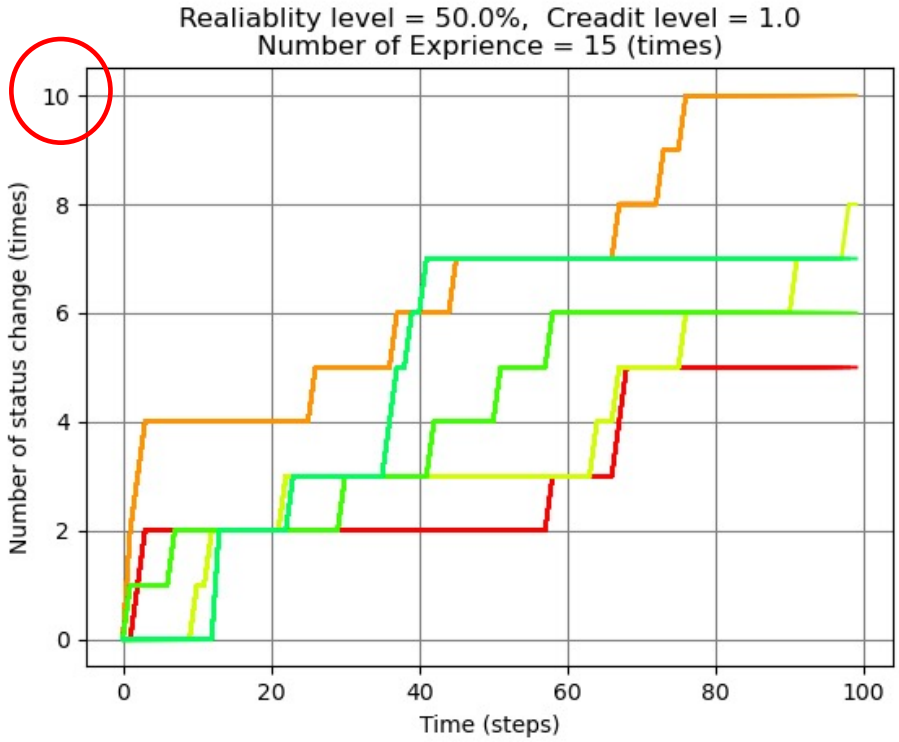


図 4-8　コンテンツの信憑性：50%、同じパーティーの信頼性：100%、実験回数を 15 回

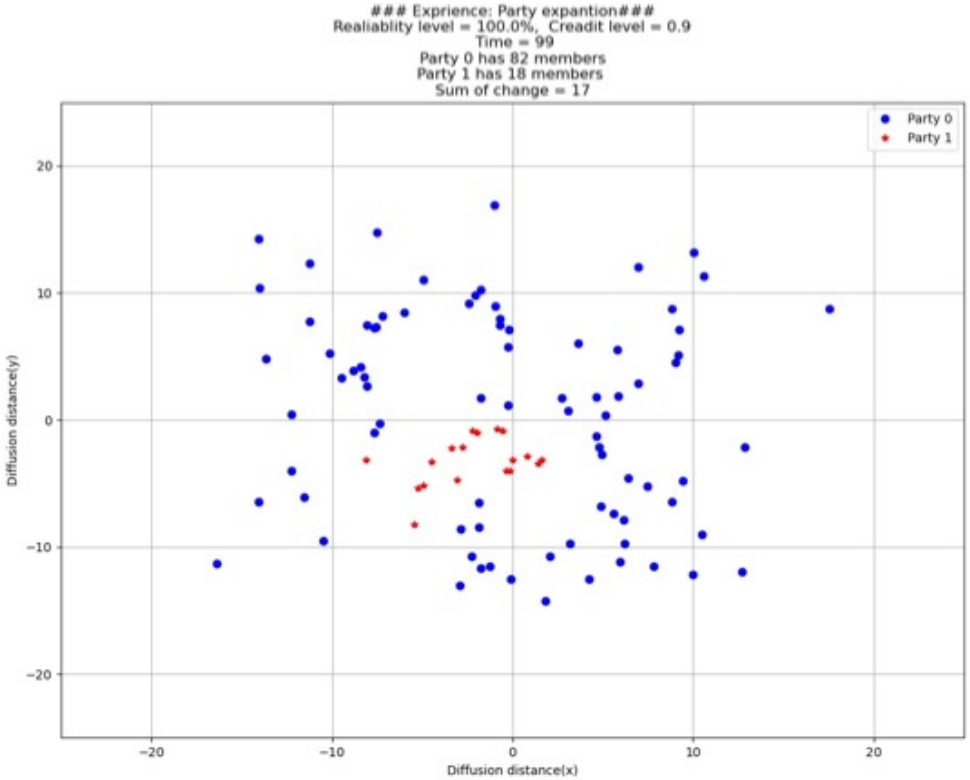


図 4-9　コンテンツの信憑性：100%、同じパーティーの信頼性：90%、N = 50 nodes、T = 100 steps　最初は Party 1 が 1 node である場合の結果の一部

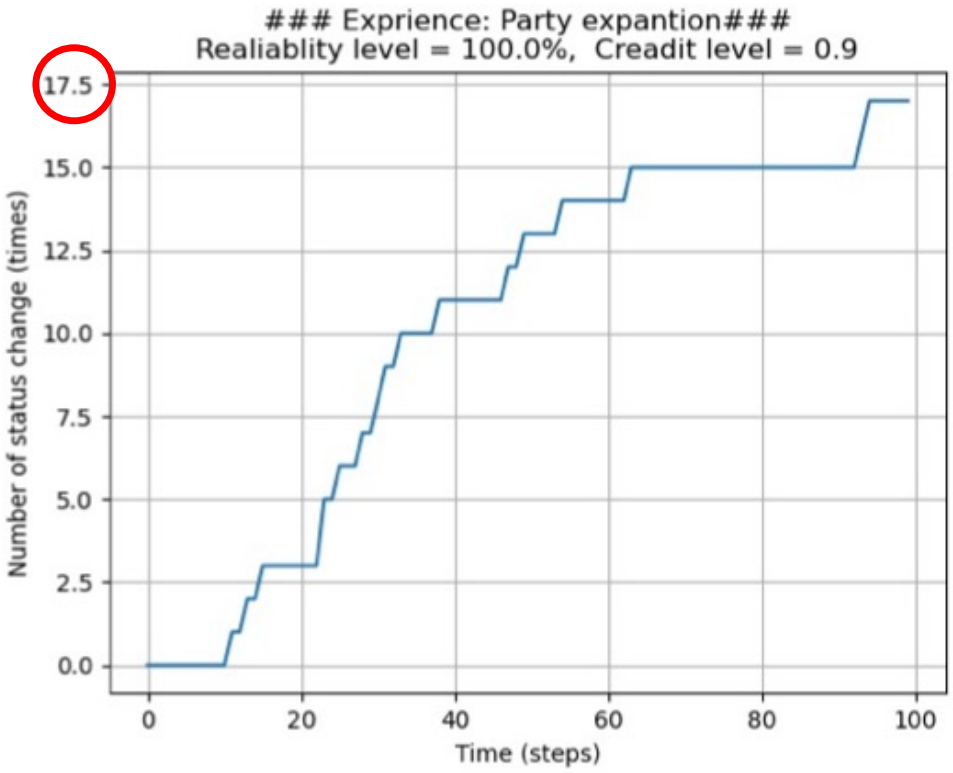


図 4-10 図 4-9 の寝返り数の時間的変化

4.5.3 影響度 R を変える

影響度 R を大きくし、影響を近接したノードから遠くにあるノードまで変化させてみる。影響度 R は群衆の中での声の大きさであると例えたように、ノードの移動は大きくない状態であっても、半径 R 内のノードに「影響」を与えることになる。

具体的に「影響」とは、あるノードが別の集団のノードを強制的に自分の集団に帰属させることである。R が大きいと自らの集団の味方を増やす頻度が増えることになる。

SNS では、R が大きいことは、その定義からリンクの成立が多いということになる。つまり、1 つのノードのメッセージがリンクによって運ばれる頻度が上がることになり、メッセージが拡散しやすくなる。

図 4-11 は、パーティー（0）もパーティー（1）も同数で対峙する時のノードの動きの中で、観測期間 T が満了（T = 100）となった時点の様子を示す。青と赤はそれぞれのノードを示す。

R を 0.01, 0.05, 0.1, 0.2, 0.3, 0.5 と小刻みに変化させた結果を図 4-11 から図 4-16 まで示した。

図 4-11 から図 4-16 までのいずれも赤のパーティー（1）の方が、青のパーティー（0）よりも多く、赤が優勢に見える。しかし、これはあくまでも偶然で、何度か実験を繰り返すと、赤が常に優勢であることはなかった。

観測期間 T が満了となった時点での寝返りに注目すると、パーティー（0）のノード数が 50 に対して、図 4-11 から図 4-16 での寝返り数はそれぞれ、8, 26, 34, 35, 36, 43 となった。R の変化に対して寝返り数は非線形で単調には増加していないことがわかった。

さらに詳しく見るために、同一条件で、$0.01 \leq R \leq 1$ で R=0.01 で小刻みに変化させた結果を図 4-17 に示す。

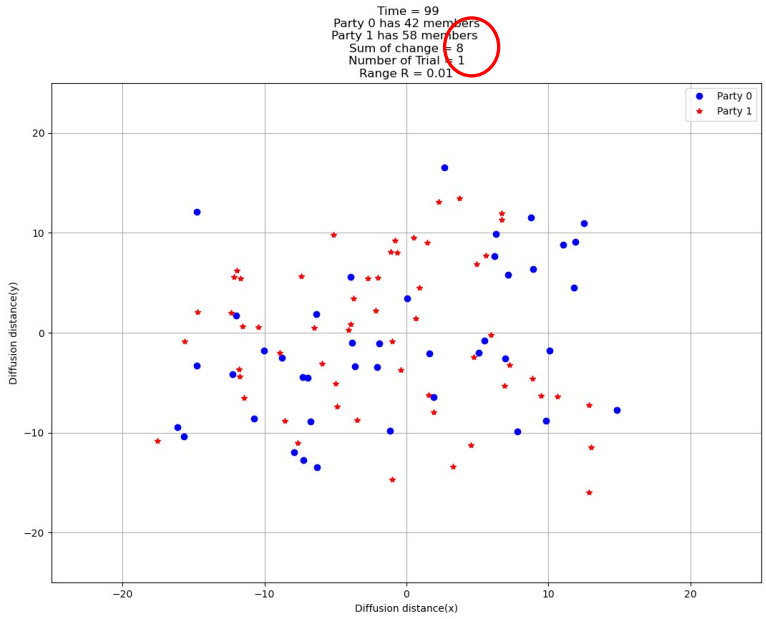


図 4-11 コンテンツの信憑性：100%、同じパーティーの信頼性：100%、N = 50 nodes、T = 100 steps、最初は Party 1 が 50 nodes、R = 0.01、8 node を寝返らせている

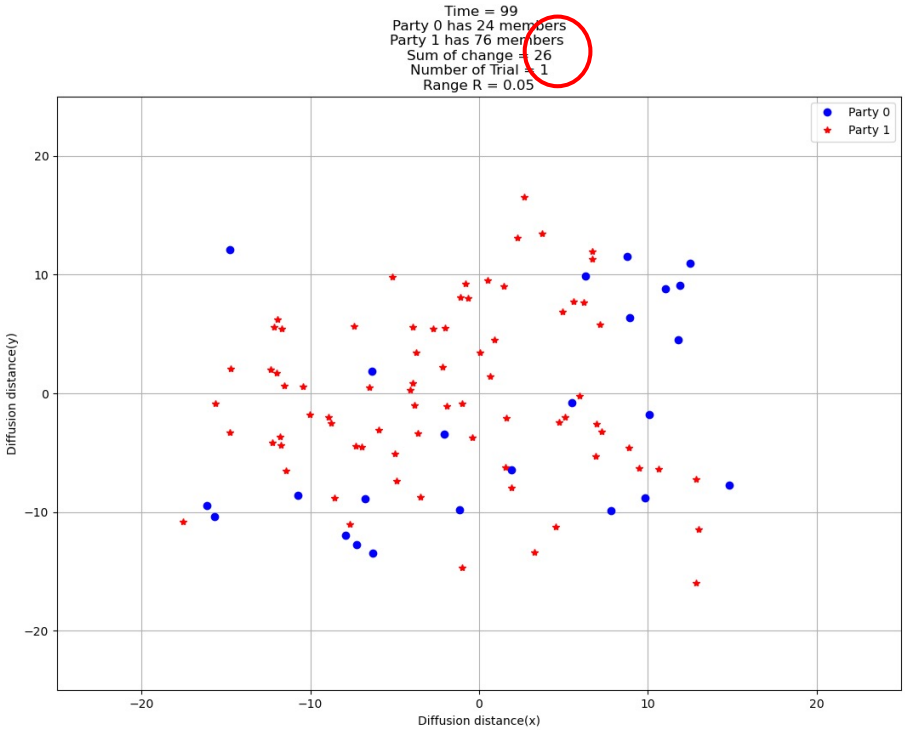


図 4-12 コンテンツの信憑性：100%、同じパーティーの信頼性：100%、N = 50 nodes、T = 100 steps、最初は Party 1 が 50 nodes、R = 0.05、26 node を寝返らせている

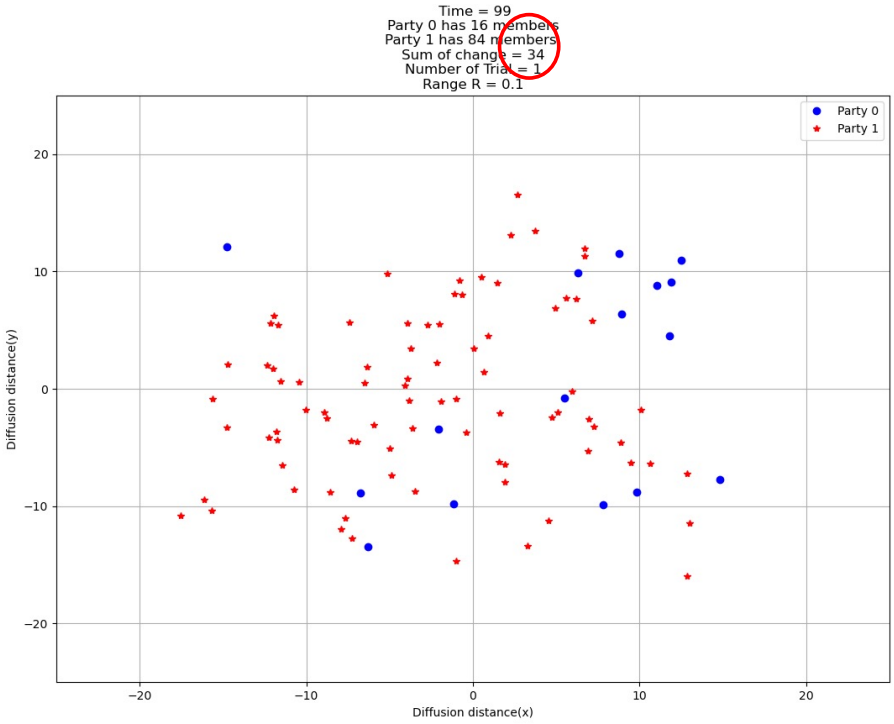


図 4-13　コンテンツの信憑性：100%、同じパーティーの信頼性：100%、N = 50 nodes、T = 100 steps、最初は Party 1 が 50 nodes、R = 0.1、34 node を寝返らせている

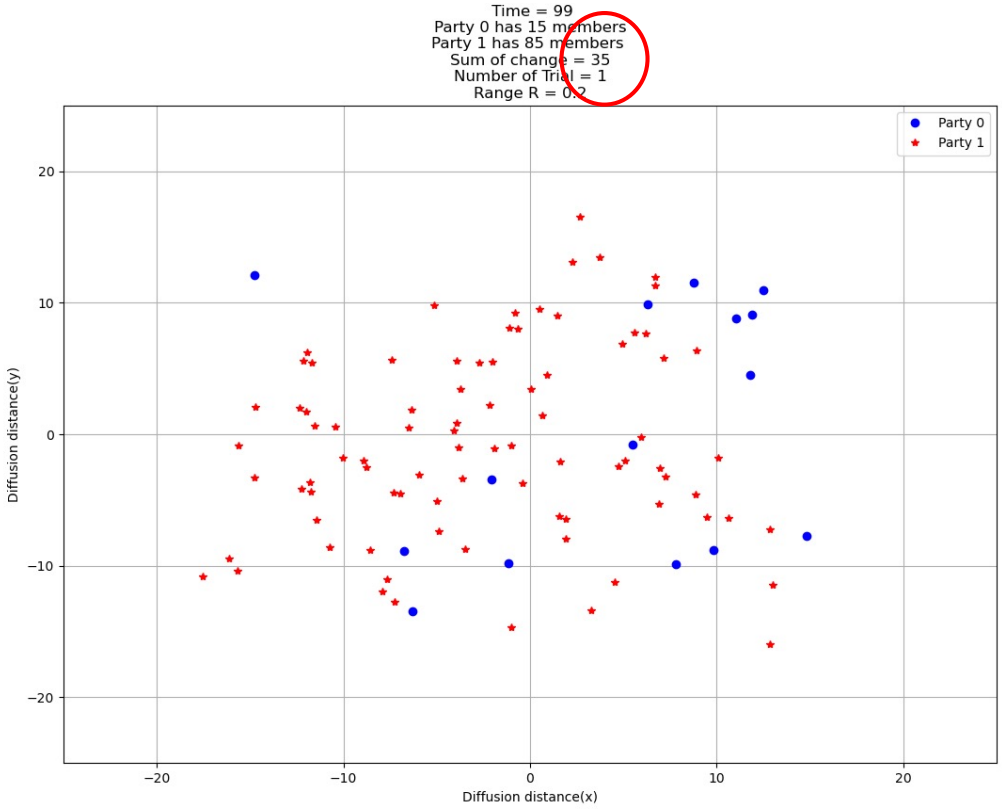


図 4-14　コンテンツの信憑性：100%、同じパーティーの信頼性：100%、N = 50 nodes、T = 100 steps、最初は Party 1 が 50 nodes、R = 0.2、35 node を寝返らせている

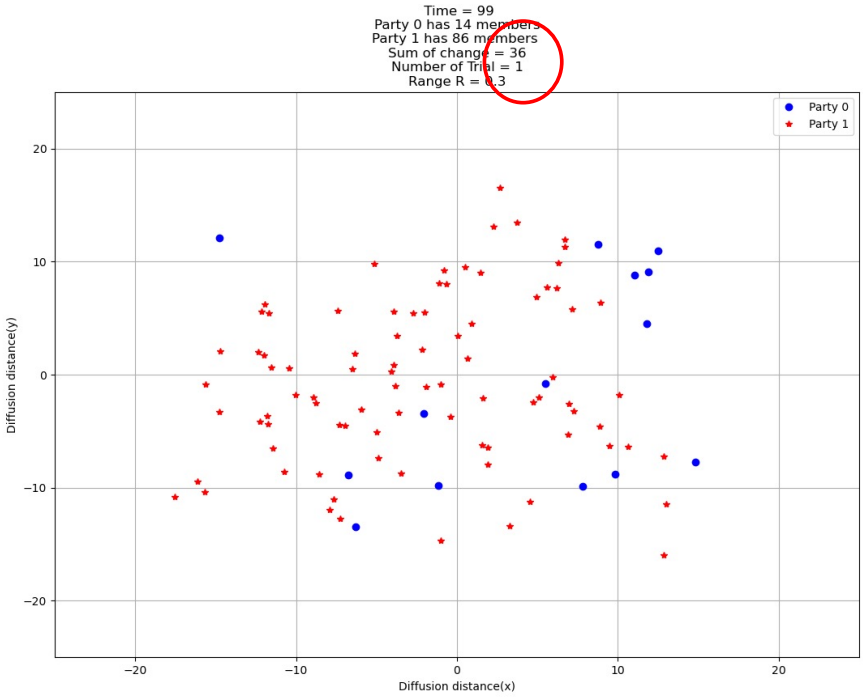


図 4-15　コンテンツの信憑性：100%、同じパーティーの信頼性：100%、N = 50 nodes、T = 100 steps、最初は Party 1 が 50 nodes、R = 0.3、36 node を寝返らせている

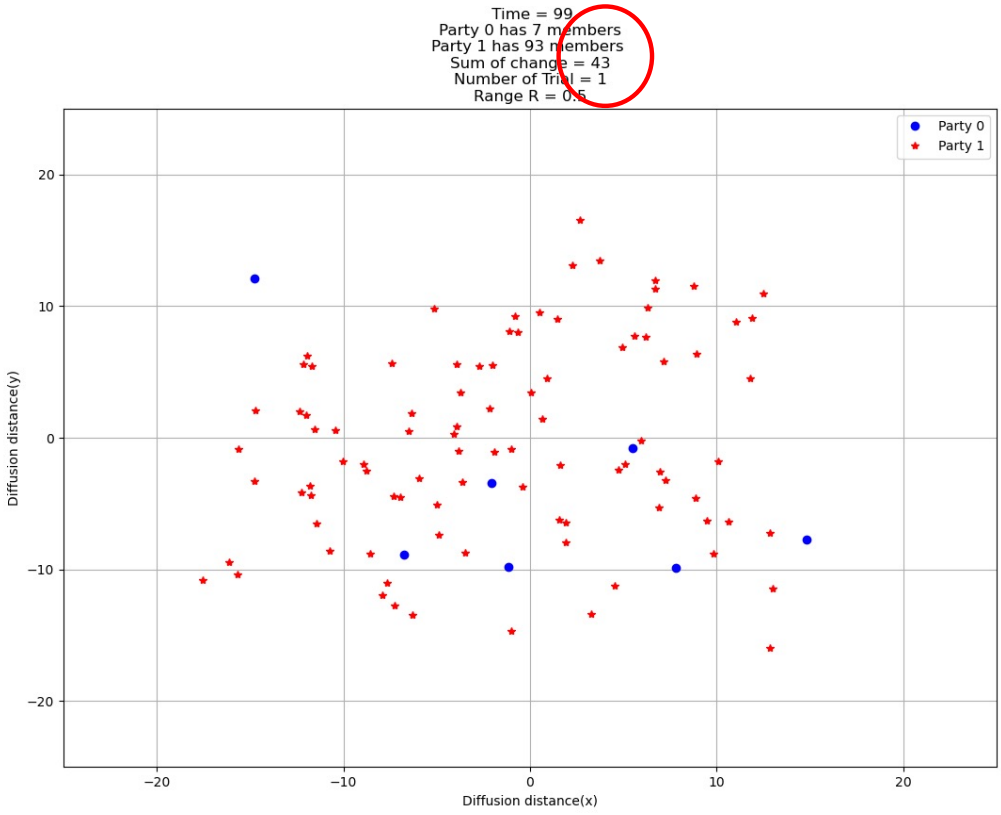


図 4-16　コンテンツの信憑性：100%、同じパーティーの信頼性：100%、N = 50 nodes、T = 100 steps、最初は Party 1 が 50 nodes、R = 0.5、43 node を寝返らせている

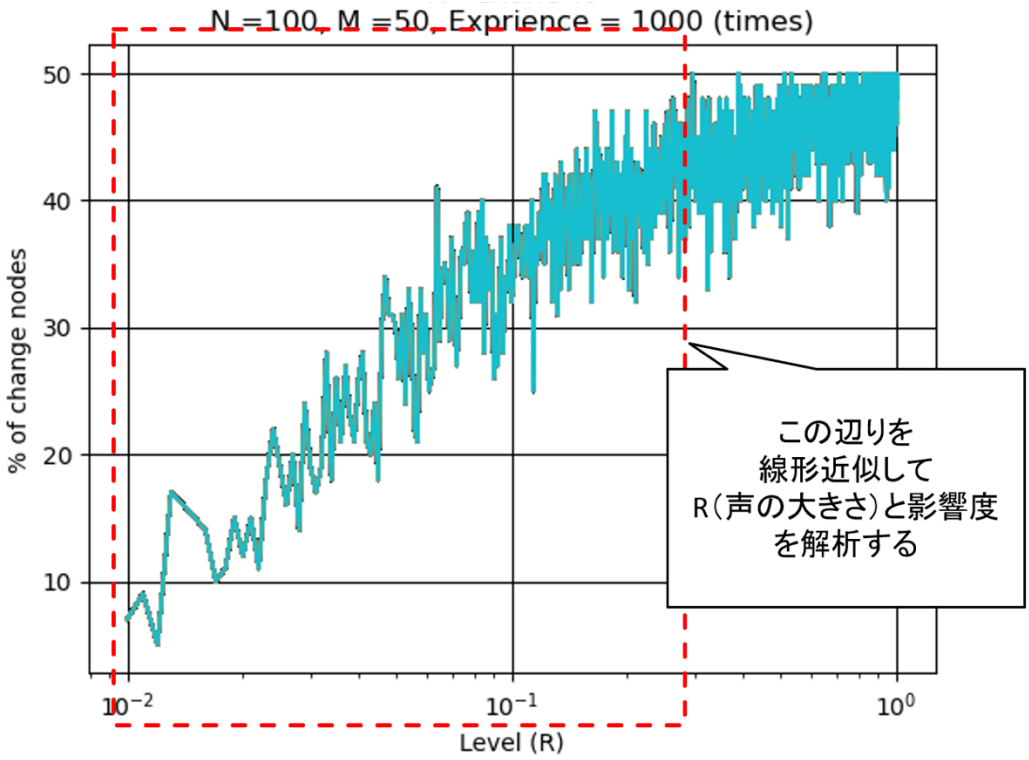


図 4-17　コンテンツの信憑性：100%、同じパーティーの信頼性：100%、N = 1000 nodes、T = 100 steps、Party 0: party 1 = 500 : 500、R の刻み = 0.01

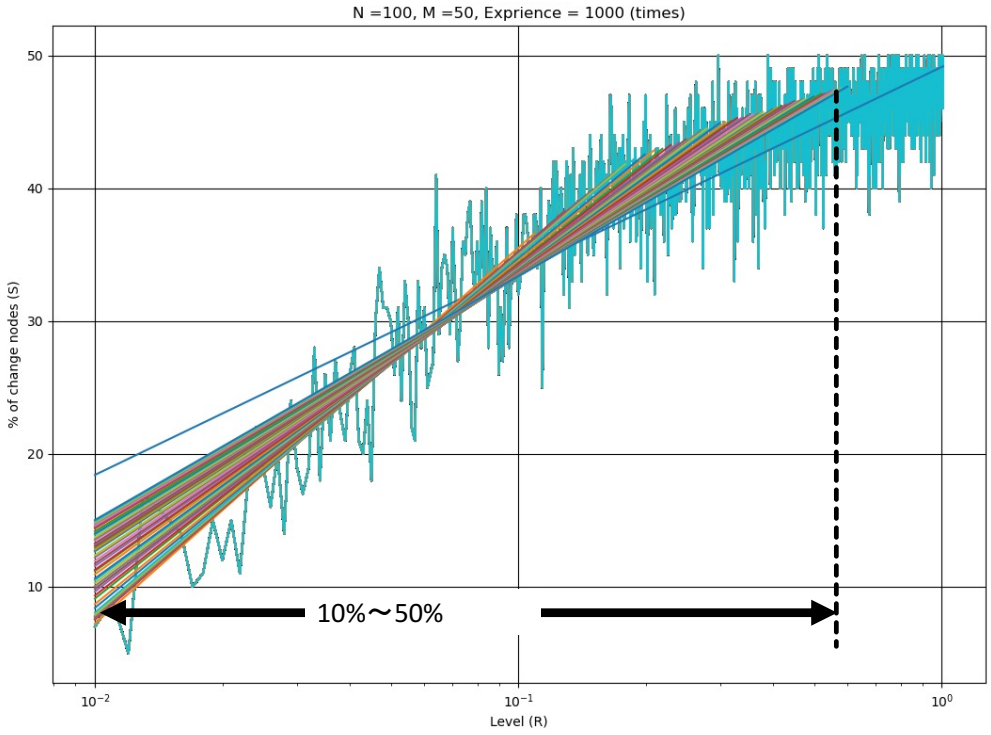


図 4-18　コンテンツの信憑性：100%、同じパーティーの信頼性：100%、N =100 nodes　T = 1000 steps、M=50 で R は対数目盛で図 4-17 を一次関数で近似。

ここまで寝返りのノード数に注目してきたが、R の影響を詳細に見るために寝返ったノード数を全ノード数で割った「寝返り率」で解析する。

さらに、R の値を変化させ、最も寝返り率が高くなる極大値を探ることにした。対数的に R を増加させると図 4-17 のように寝返り率も増加する。同図の波線で示した部分は線形関数的であるので、これを近似する。図 4-17 の波線の範囲を窓と呼んで、窓内を図 4-18 のように 1 次関数(7)式の係数 a と b を変えて最も相関係数が高いものを選ぶことにした。

$$C = a + b * ln\, R \tag{7}$$

ここで、C は寝返り率、R は影響度である。縦軸を(7)式の決定係数とし、図 4-17 の窓のサイズを横軸とすると、図 4-19 の分布となった。ここから、R に対して窓内で最も寝返り率 C が極大となるのは C = 0.8 で、(7)式から逆算して a = 56.2、b = 22.83 で極大となる。 影響度 R が大きくなると一様に集団が大きくなるのではなく、(7)式で示したように極めて小さい範囲で留まっている。（この実験では、窓の範囲が約 8%で決定係数が約 0.8 で極大となった。）それ以上Ｒが窓の範囲を超えると、同じ集団のノード数が膨張することがわかる。

このように、影響度 R が小さい集団では、自分の所属する集団を拡大することは難しく、一定の R 以上になると、他の集団のノードを寝返らせる方が多くなり、集団が拡大していくことが、図 4-17 の観測時間を 1 から最終ステップまで連続的に観測するとわかる（参照：付録３：第４章のシミュレーションの動画結果）。

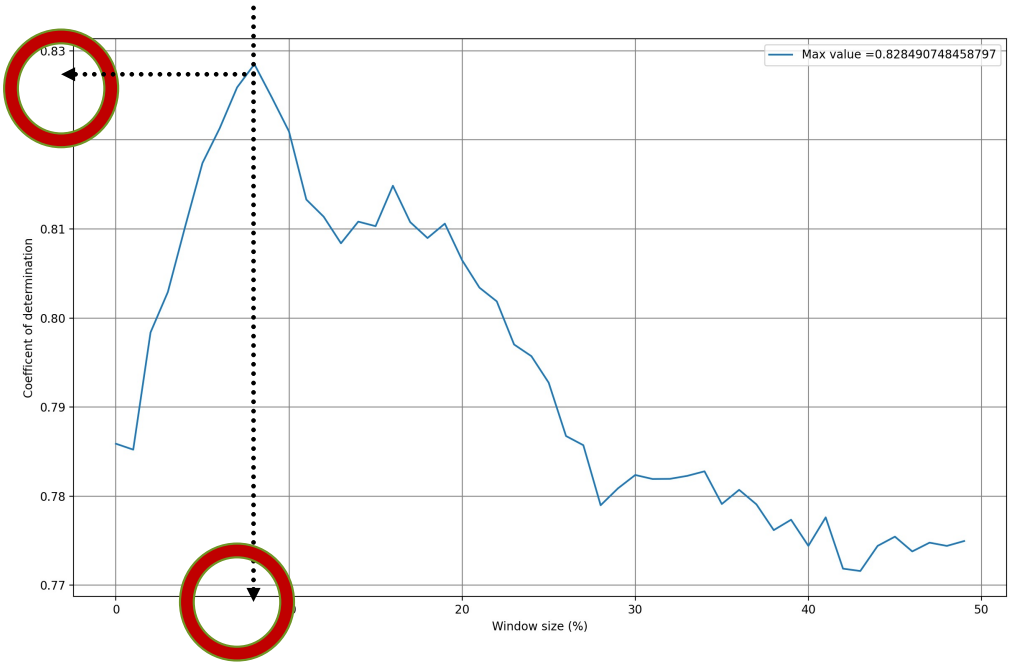


図 4-19　図 4-17 の近似係数　図 4-17 の最小から約 0.8 のサンプル数で決定係数が最大になる

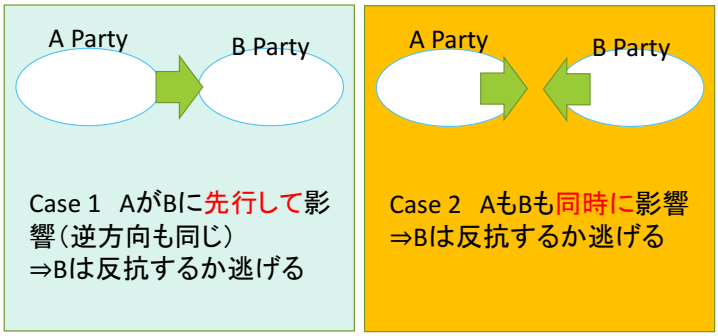


図　4-20　同数のノードを持つ集団 A と集団 B と影響力発揮のタイミング

ノード数の拮抗する２集団がメッセージ空間で接触する場合、相手に対して影響を与えるタイミングで動態が変わるように思われる。図 4-20 に示すように、結果として対抗するパーティー（B）は、パーティー(A) にあくまでも反抗するか、逃げる（接触を避ける）しか選択がないためにタイミングには依存しない。

依存するのは、タイミングではなく、集団内でのノードの影響である。このシミュレーションの例で考えてみよう。

寝返りを集団戦略としているので各ノードは 3 つの影響の仕方があり、

① どちらの集団に帰属しても、相手を寝返りさせる
② どちらの集団に帰属しても、相手に吸収される（帰属させられる）
③ どちらの集団に帰属しても、影響度の範囲から出ることで逃げる

のいずれかの選択をとることになる。

影響度 R と各集団に属するノード数によって①から③の何かを選択することになる。例えば、①、②、③にノードの 40%、20%, 40%で分布した時とノードの 20%、40%, 40%で分布した時では、前者は攻撃型で相手を自分の集団に取り込む影響力が大きい。後者は、自分の集団を相手から影響を受けても生き残ることを優先するサバイバル型になる。

実際、SNS で影響力を持つためには、攻撃型かサバイバル型か何れがよいのか。

今回のシミュレーションでは、これらの分布はランダムとしたので、攻撃型やサバイバル型を設定はしていない。ただ、本研究の発展では取り上げられる内容である。

4.6　第 3 集団の動態

これまでは１つの集団に別のノードあるいは同じ意見を持つ集団が接近した場合メッセージの広がる様子がわかった。さらに 2 つのメッセージ集団に第 3 のノード、あるいはその第 3 ノードが属する集団が接触した場合を考える。この時、各集団が無反応な場合や第３集団を無視する場合を除き、図 4-21 のように４つのパターンが考えられる。

図 4-21 の集団 A、B、C について、

① Case 1: 集団 C が集団 A、集団 B の両方から影響を受ける場合
② Case 2: 集団 C は集団 A からは影響を受け、集団 B に影響を与える場合
③ Case 3: 集団 C は集団 B からは影響を受け、集団 A に影響を与える場合
④ Case 4: 集団 C が集団 A、集団 B の両方に影響を与える場合

である。

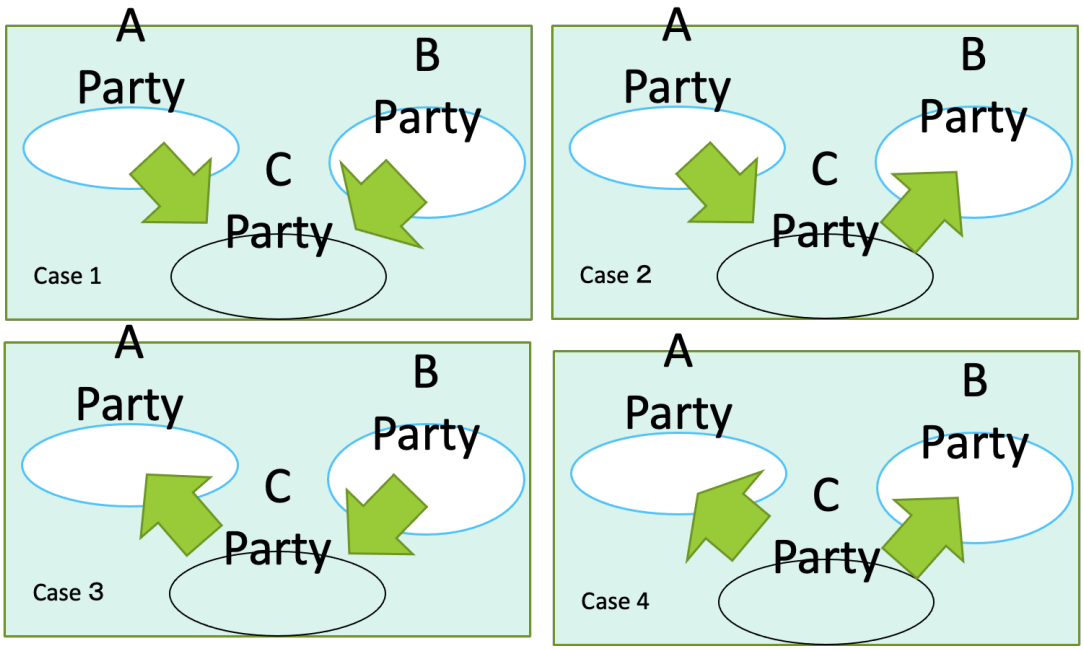


図 4-21 集団 A, 集団 B と第三集団 C

①は、少なくても集団 A と集団 B の両方から影響を受け、集団 C のメッセージはかき消される可能性がある。②は、集団 A からのメッセージでかき消される可能性があるが、集団 B で集団 C のメッセージが残る可能性がある。③は②と相手が変わり、集団 B からのメッセージでかき消される可能性があるが、集団 A で集団 C のメッセージが残る可能性がある。④では、集団 A あるいは集団 B で集団 C のメッセージが残る可能性がある。

4.6.1 同数の集団に対する第 3 集団

同数で対峙する２集団のシミュレーションで使ったモデルに第３の集団 C のノードを入れて観察してみる。狙いは、第３集団が他の２集団にどのような影響を与えるのかを観測することである。

図 4-22 は、第 3 の集団をパーティー（２）(緑)として、既存のパーティー（０）(青)、パーティー（１）(赤)に侵入した場合である。なお、パーティー（２）(緑)は集団といっても１ノードのみで、他の集団は 50 ノードずつで同数である。これは、パーティー（０）(青)、パーティー（１）(赤)が同数で張り合っているなかで、１ノードのパーティー（２）(緑)が侵入してきた状況である。

なお、0 ステップでパーティー（０）(青)とパーティー（１）(赤)が同数となっていないのは、初期値で既に相手に寝返りが起きているためである。

計算の結果を見ると、

- 青と赤の線がステップ 1 ですでに青は赤を寝返らせている。緑は生き残っている。この時点までは３集団が残っている。
- 青と赤の線がステップ 5 あたりで同数となり、その時、緑は生き残っている：この時点までは３集団が残っている。
- 23 ステップで緑は消滅し、同時に青が赤より優勢となっている：青が緑を吸収し、赤よりも優位に立っている。
- 以後、青が赤よりも優勢で、緑は再生しない：

形勢を変える鍵となる緑がないため逆転しない。
となった。少数である第３のパーティー（２）(緑)が、上記①のパターンで消滅した。これは、パーティー（２）が、多数決の原理で少数派が決定権をもつキャスティングボートを握るような立場になった例である。

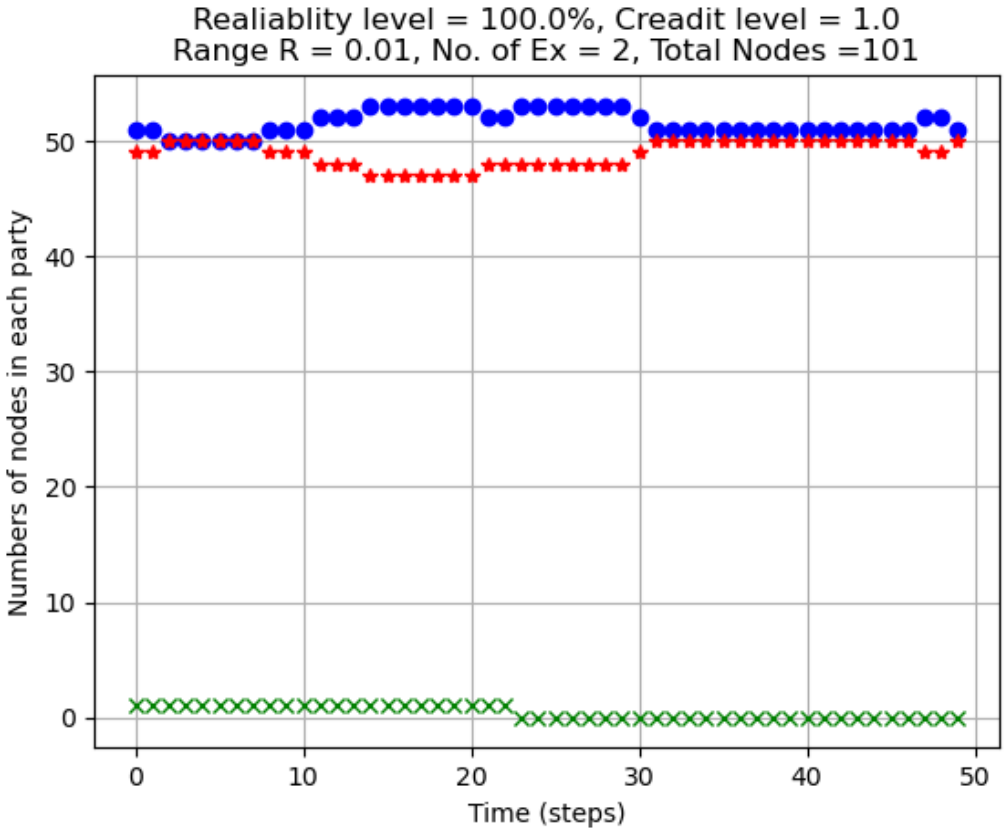


図 4-22　コンテンツの信憑性：100%、同じパーティーの信頼性：100%で R=0.01 の例。パーティー0 が青、パーティー1 が赤、第 3 のパーティー2 が緑である。

図 4-23 は、図 4-22 と同じ条件で、第３集団（１ノード）をパーティー（２）(緑)として、既存のパーティー（０）(青)、パーティー（１）(赤)に侵入した場合である。なお、パーティー（２）(緑)は１ノードで、他の集団は 50 ノードずつの同数である。
計算の結果を見ると、

- 青と赤の線がステップ 6 で同数となり拮抗するが、8 ステップ以降は青が他よりも常に優勢となる：
  この時点までは３集団が残っている。
- 20 ステップで緑は消滅し、同時に青はさらに赤より優勢となっている：
  青が緑を吸収し、赤よりも優位に立っている。
- 38 ステップ以後、青が赤よりも常に優勢で、緑は再生しない：
  形勢を変える鍵となる緑がないため逆転しない。

となった。ステップ 6 でパーティー（０）(青)はキャスティングボートを握るパーティー（１）(赤)を取り込んで一気に優勢に立った結果である。

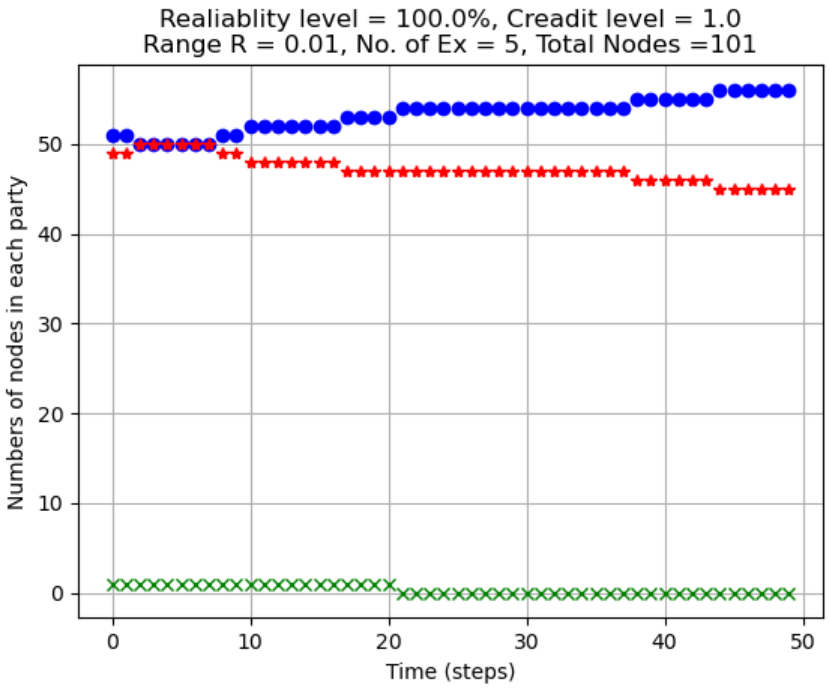


図 4-23　コンテンツの信憑性：100%、同じパーティーの信頼性：100%で R=0.01 の例で第 3 集団を飲み込んだ

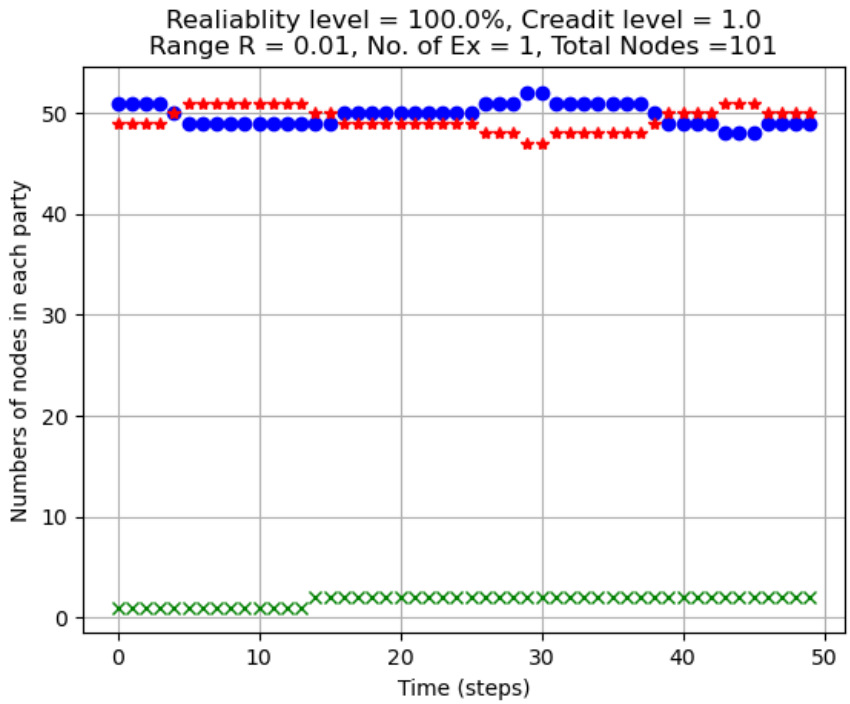


図 4-24　コンテンツの信憑性：100%、同じパーティーの信頼性：100%で R=0.01 の例で第 3 集団は生き残っている。

図 4-24 も、図 4-22 と同じ条件で、第 3 集団（1 ノード）をパーティー（2）(緑)として、既存のパーティー（0）(青)とパーティー（1）(赤)に侵入した場合である。

計算の結果を見ると、

- 青と赤についてステップ６で赤が青より優勢となり、ステップ 16 で青と赤が逆転、ステップ 38 で再び逆転して最後は赤が優勢となっている：
  最後まで３集団が残っているが、赤と青のどちらが優勢か決まらない。
- 緑はステップ 16 の青と赤の逆転に際して、ノード数を 1 から２に増やしている。この増分は、初期では青と赤のノード数は同じであることから赤を緑に寝返らせてノードを増やしたものと思われる。上述の②または③のケースで、第３集団が他の集団に影響を与えた例である。
- 赤は青よりステップ 16 で緑への寝返ったノードのために劣勢になるが、再びステップ 38 で、青より優位になっている。この時点の赤の優位性は、青のノードを赤に直接寝返らせ、緑には接触していない。

となった。この結果は、既存集団同士で優位性を争っており、第３集団はその合間をぬってノードを増やそうとしている。
このように、例え少数であっても第３集団は、同数の拮抗する集団に割り込むことで、優位性なる結果が観測できた。

4.6.2　同数の集団に対する第 3 のノードの R の影響

次に、対抗している同数の集団に影響範囲を変えて第３集団が侵入するとどうなるかを観測する。この観測の狙いは、影響範囲がノード数の優位性にどう関係しているのかをみることである。
図 ４-25 は、R = 0.1 で第 3 の集団（１ノード）をパーティー（２）(緑)として、既存のパーティー（０）(青)とパーティー（１）(赤)に侵入した場合である。
この計算の結果を見ると、

- 青と赤について、常に青が赤よりも優勢でステップが増えるほど、差が大きくなっている。影響範囲が大きくなったために、接触頻度が大きくなって、動きが図 ４-24 よりも大きくなっている
- 緑はステップ 11 で消滅し、急激に優勢となった青に取り込まれたと思われる。上述の②または③のケースで、第３集団が他の集団に影響を与えた例である

となった。同じ条件で、別の結果が図 ４-26 である。図 ４-26 を図 ４-25 と比較すると、

- 図 ４-24 とは逆に青と赤については、ステップ 3 で急激に赤が大きくなり、ステップ 5 前後では青よりも赤が優位になり逆転している。
- 緑はステップ 3 で消滅し、急激に優勢となった赤に取り込まれたと思われる。上述の②または③のケースである。
- 緑が消滅後も、青と赤の綱引きが続くが、青が優勢になっている。
- 緑の消滅後、綱引きは同数の 50 で対称となり、最大８ノードの寝返り分が振動しているように見える。つまり、寝返りが青と赤の間で交互に行われ、図 ４-26 のように振動となる。この振動現象はノードの総数が 101 ノードと変わらないため、自己の集団を優位にするためには、まず第３集団を吸収することである。その後第３集団が吸収されて消滅すれば、相手の集団を取り込もうとする。この時、吸収した第３集団のノード数の優位性を使う。

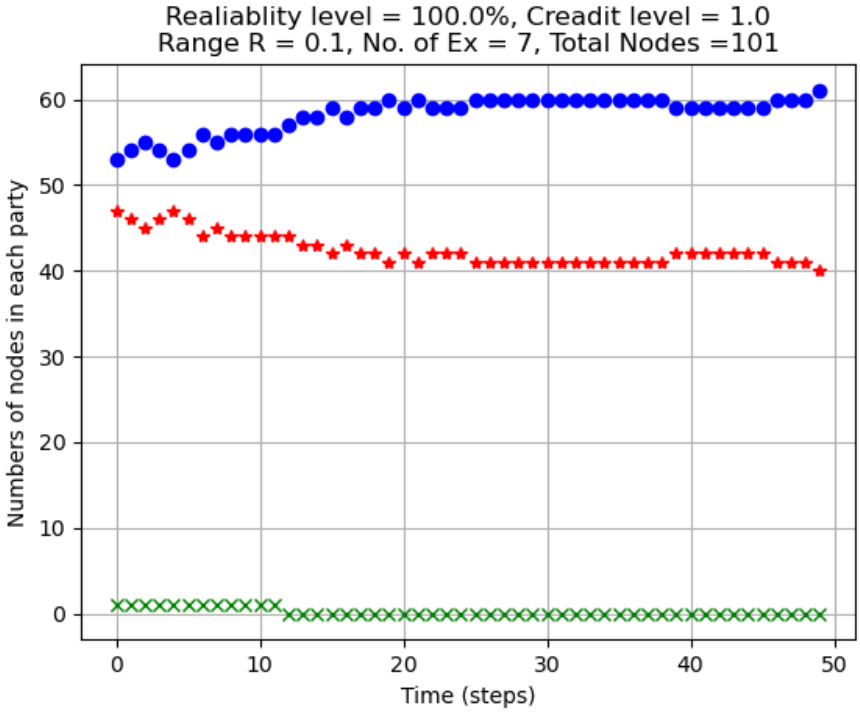


図 4-25　コンテンツの信憑性：100%、同じパーティーの信頼性：100%で R=0.1 の例で影響範囲を変えた

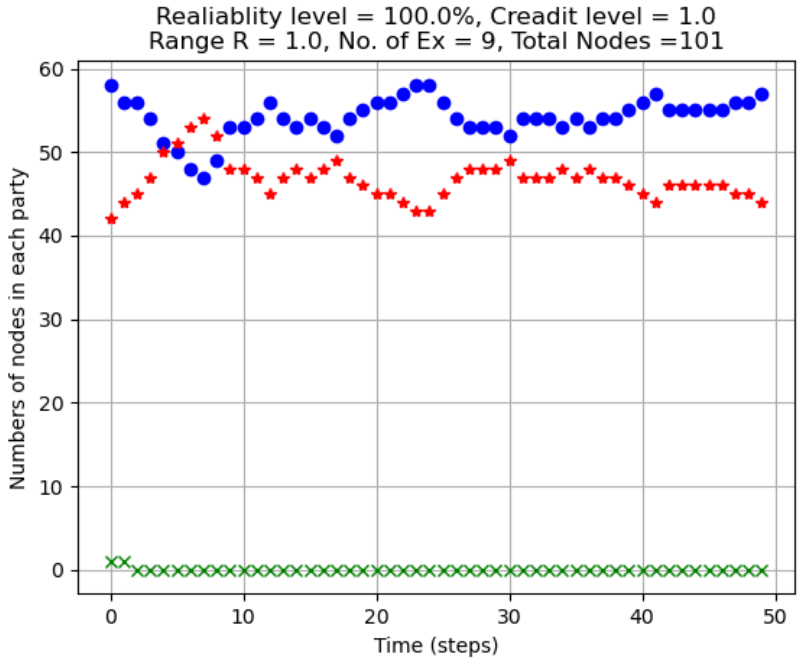


図 4-26　コンテンツの信憑性：100%、同じパーティーの信頼性：100%で R=0.1 の例

図 4-27 は、R = 1.0、第 3 集団（1 ノード）をパーティー（2）(緑)として、既存のパーティー（0）(青)とパーティー（1）(赤)に侵入した場合である。

この計算の結果を見ると、

- 青と赤について、常に青が赤よりも優勢であるが、ステップ 5 では拮抗している。

- ３集団とも観測期間内で消滅せずに残っている。
- 影響範囲が大きくなったために図 ４-26 とは異なり、振れ幅も大きくなっている。
- 緑は消滅することなく存在し、青は、１ノードから最大ノードをもつ集団となっている。

図 ４-26 と同様に青と赤の綱引きは同数の 50 で対称となっており、最大 16 ノードの寝返り分が振動していて、図 ４-26 の時よりも R の増加で寝返りの振動幅が大きい。

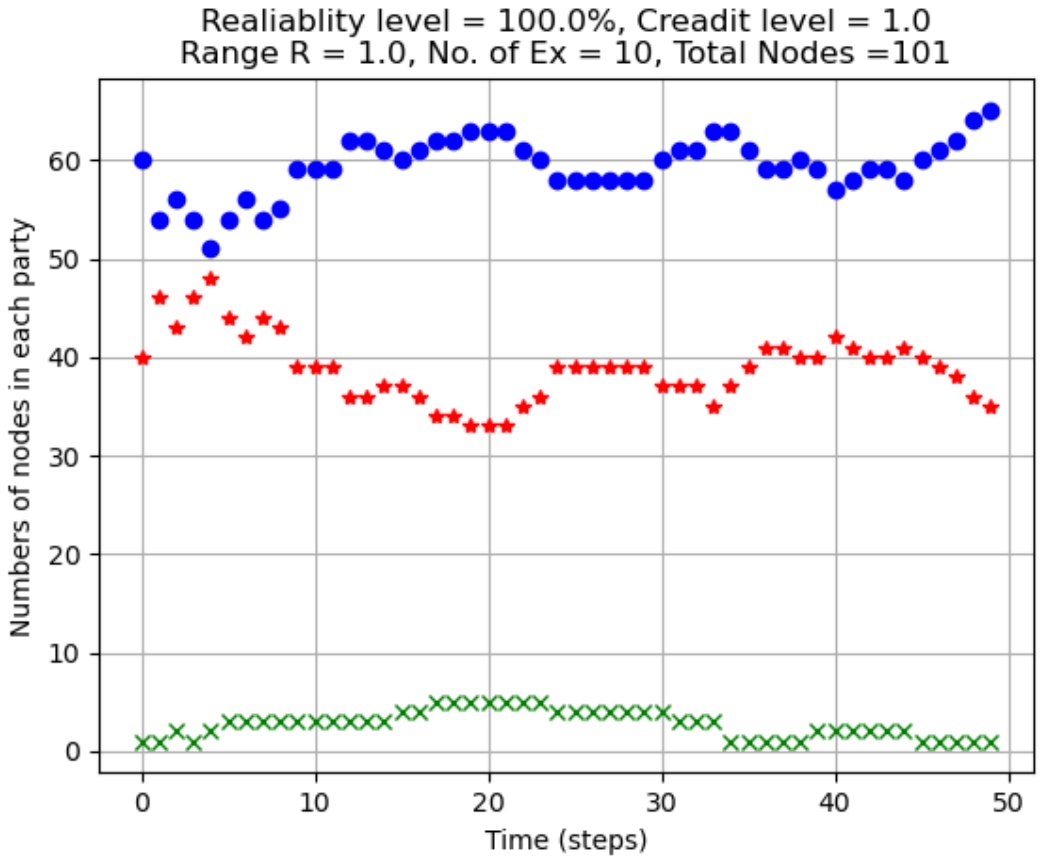


図 ４-27　コンテンツの信憑性：100%、同じパーティーの信頼性：100%で R=1.0 の例

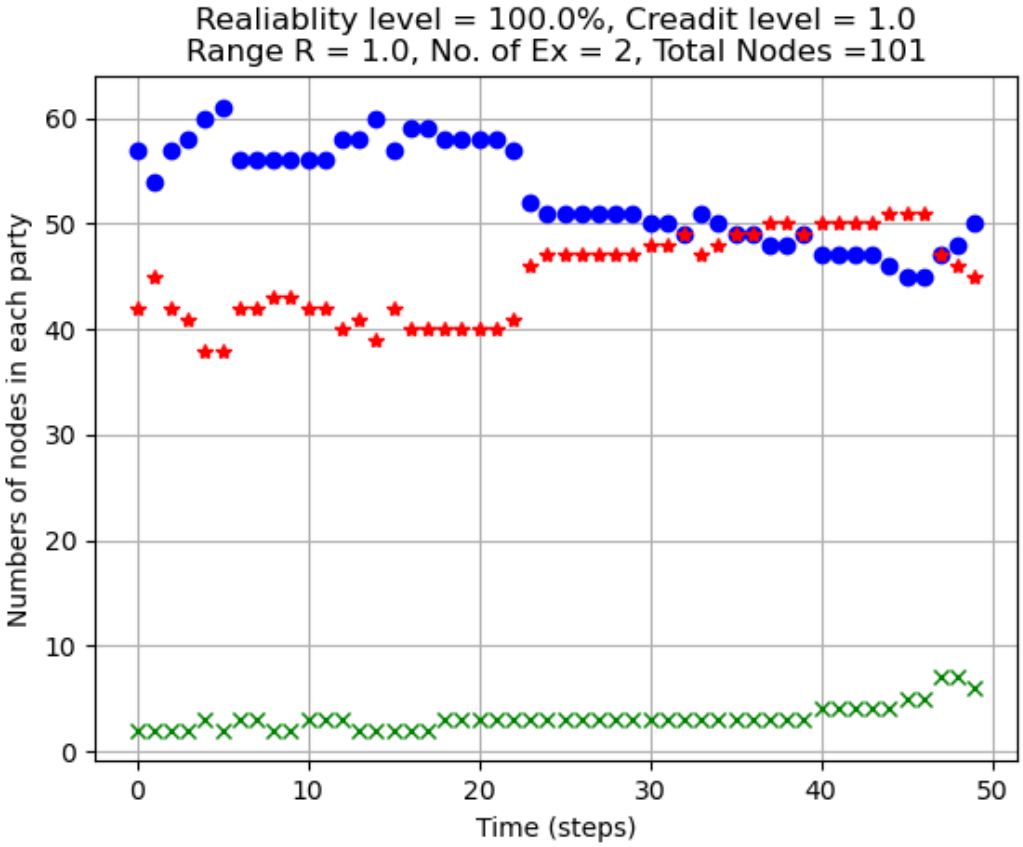


図 4-28　コンテンツの信頼性：100%、同じパーティーの信頼性：100%で R=1.0 の例

図 4-28 は、図 4-25 と同様で R = 1.0、第３集団（１ノード）をパーティー（２）(緑)として、既存のパーティー（０）(青)とパーティー（１）(赤)に侵入した場合である。

この計算の結果を図 4-27 と比較すると

- 青と赤について、優位性がステップ 32 とステップ 47 で 2 回反転している。
- ３集団とも観測期間内で消滅せずに残っているところは同じである。
- 振れ幅は最大 12 と小さい
- 緑は消滅することなく存在し、ステップ 17 から 24 では、6 ノードをもつ集団となっている。
- 緑と赤のノード数の増減が似ている。赤が優位になるとき緑も増加していることから、赤は緑を取り込むのではなく、青を寝返らせて赤に変え、青の優位性を落とし、その間に緑は現状を維持している。

この結果は、勢力分布が単純な弱肉強食で決まるのではなく、サバイバルするために、他の集団が影響をお互いに与えている隙に、自分で足場を固めているように見える。

ここまでは、R を変化させた時の各集団の優位性について観察した。さらに R を一定にして、繰り返し計算することで、各集団の特徴がないかを観察してみる。

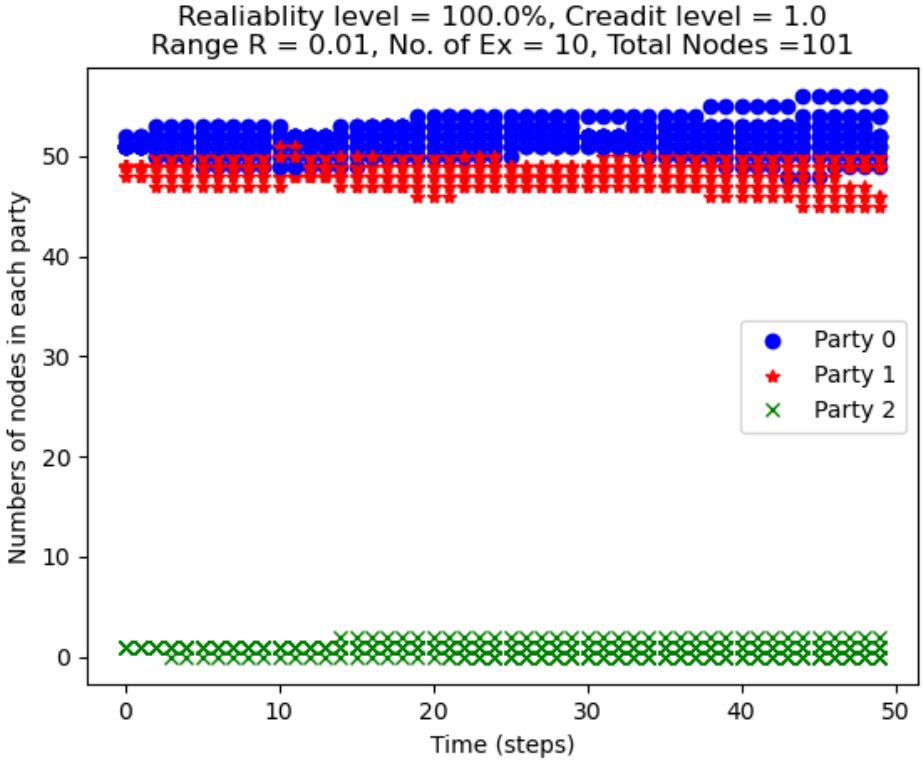


図 4-29　コンテンツの信憑性：100%、同じパーティーの信頼性：100%で R=0.01。10 回計算

図 4-29 は、図 4-24 と同様で R = 0.01、第３集団（１ノード）をパーティー（２）(緑)として、既存のパーティー（０）(青)とパーティー（１）(赤)に侵入した場合である。計算は 10 回、毎回初期化して行った。

この計算の結果をみると、

- 青と赤について、青の方が全体として優位である。
- ３集団とも観測期間内で消滅せずに残っている。
- 振れ幅は最大 5。
- 緑は最大 3 ノードをもつ集団になる場合もある。

同図は、R = 0.01 の影響範囲では、接触が少なく、集団間の変化も小さいため、グラフは、大きな振動を持たず平坦に近い。

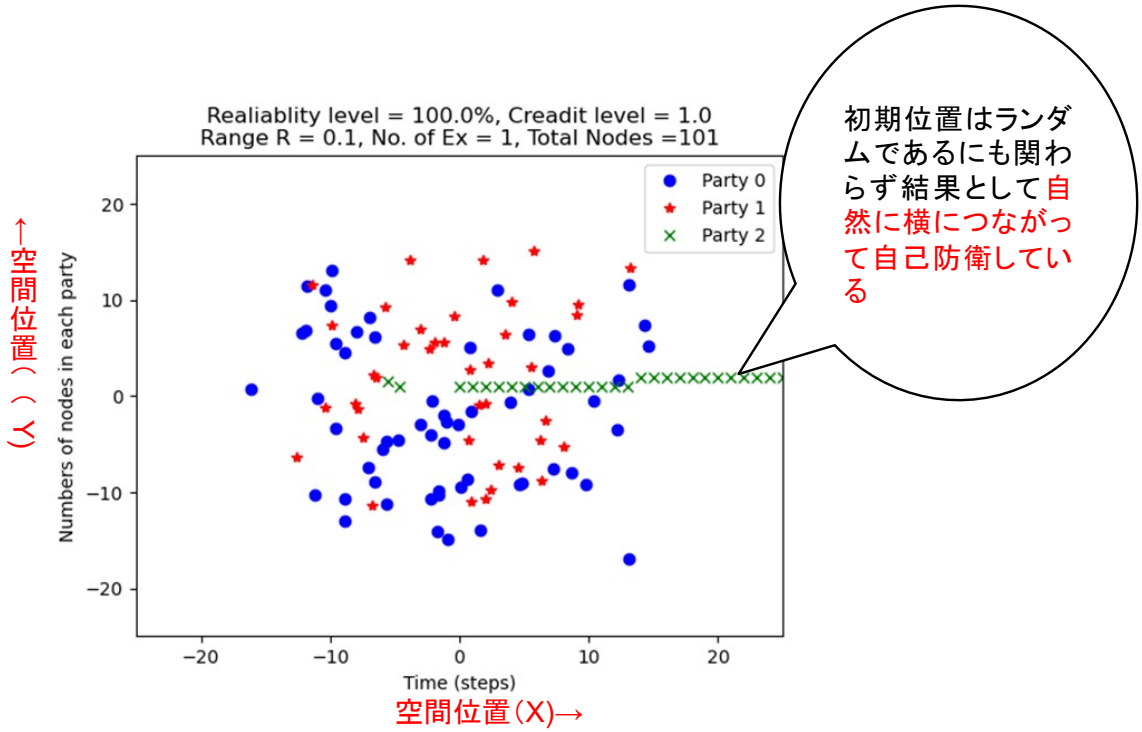


図 4-30 コンテンツの信憑性：100%、同じパーティーの信頼性：100%で R=0.1 のノード分布の一例

この時のノードの分布は図 4-30 のように第３集団は横につながって自己防衛しているような事例もある。興味深いことに、この自己防衛は自然に初期位置で発生した例である。
さらに、R = 0.1 にすると、図 4-31 の結果となった。

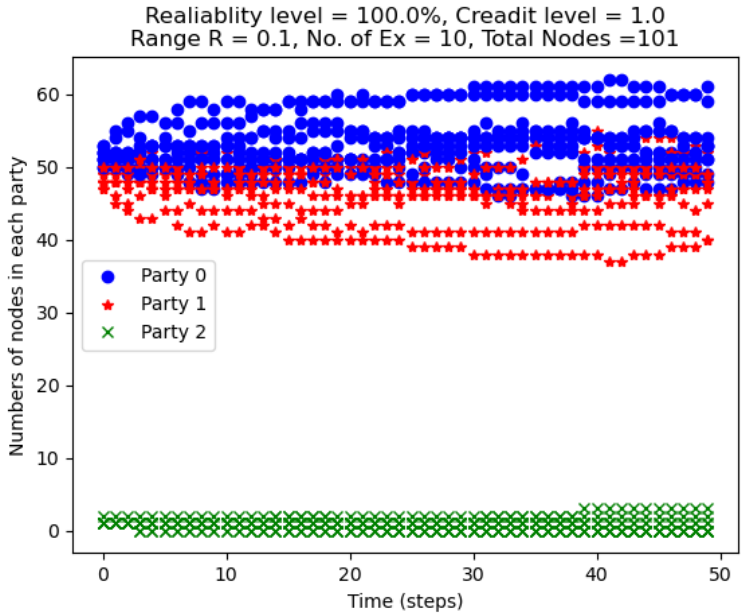


図 4-31 コンテンツの信憑性：100%、同じパーティーの信頼性：100%で R=0.1。10 回計算

この計算の結果は、R=0.01 と比較すると、

- 青と赤について、青の方が全体として優位であるが、赤が優位になることもある。
- ３集団とも観測期間内で消滅せずに残るのは、緑が増加した時である。
- 振れ幅は最大 12 で増加。
- 緑は最大 3 ノードをもつ集団になる場合もあるが、多くが他の集団に吸収されていく。

図 ４-31 の R = 0.1 の影響範囲では、接触が増え、振動や寝返りが頻繁に起こっている。

さらに、R = 1.0 を見てみよう。

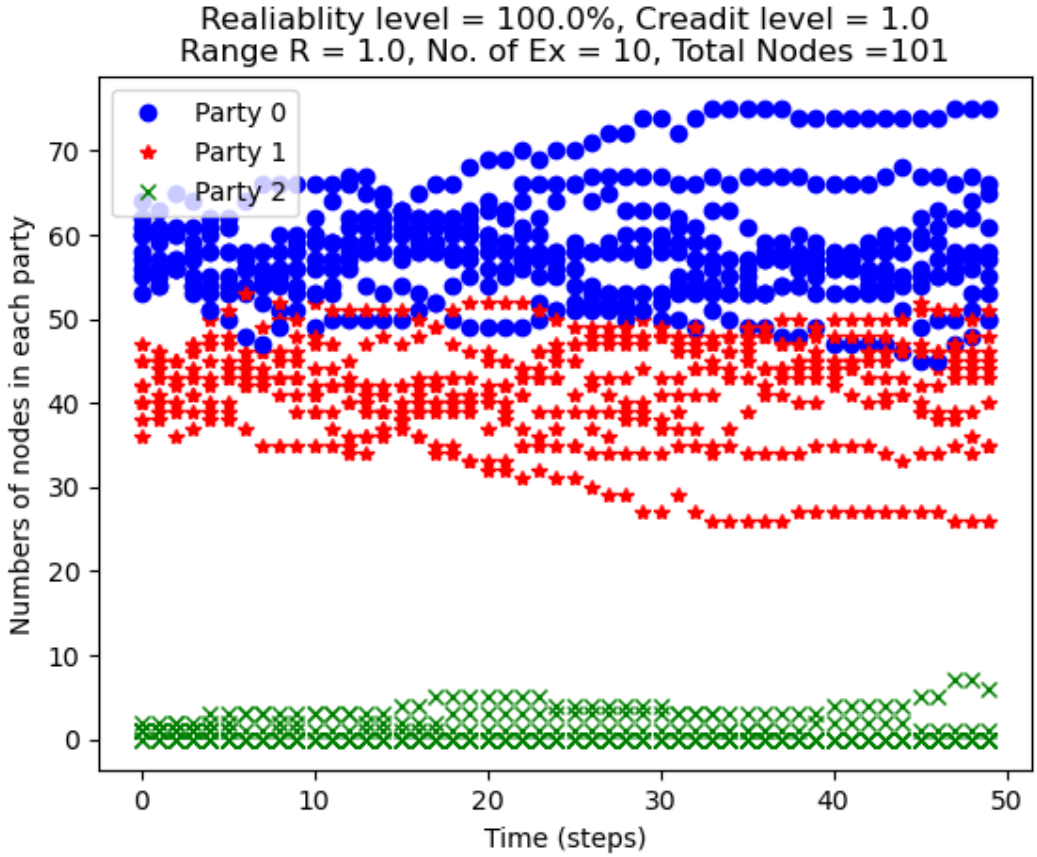


図 ４-32　コンテンツの信憑性：100%、同じパーティーの信頼性：100%で R=1.0。10 回計算

この計算の結果は、R=0.1 と比較すると、図 ４-32 から

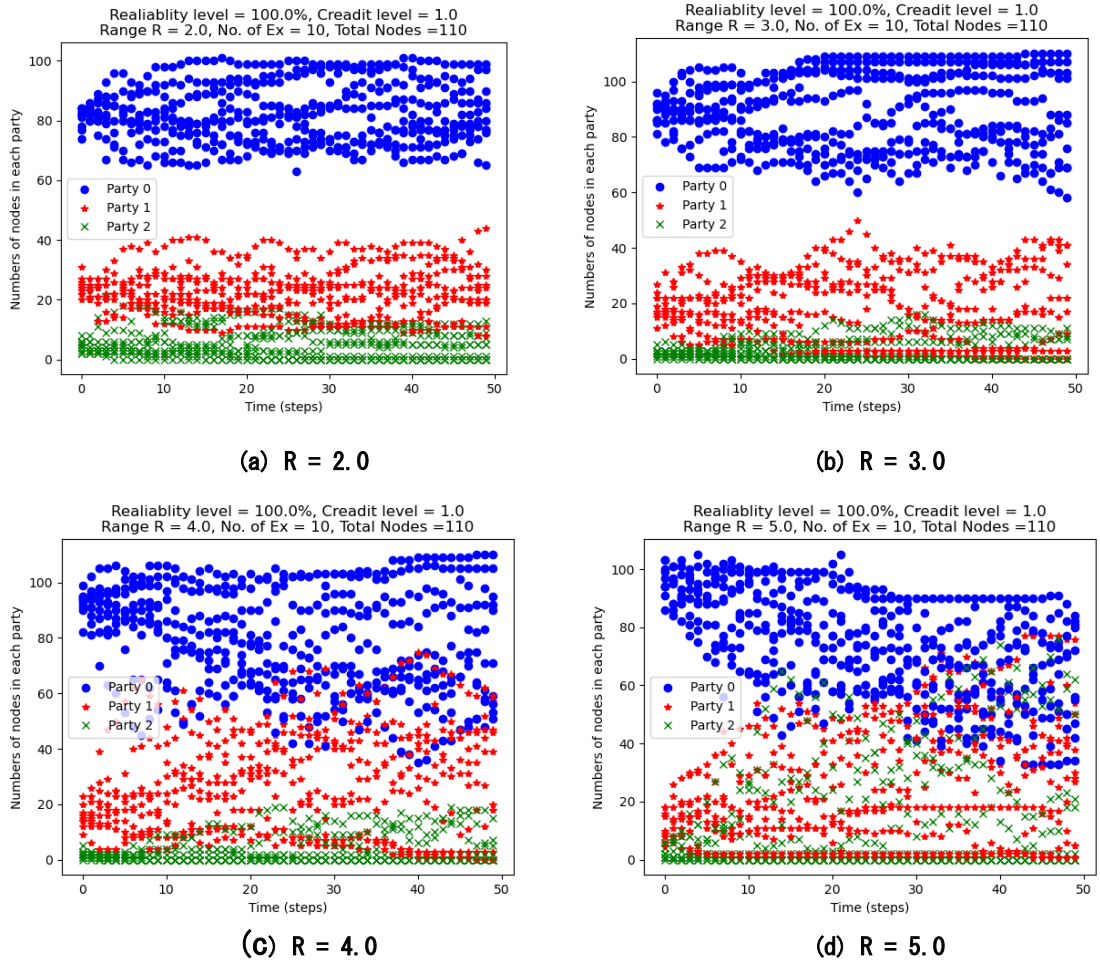
- 青と赤について、ノード数 50 を見ていくと、青と赤の優劣は同程度で分布している。
- ３集団とも観測期間内で消滅せずに残るのは、緑が増加した時で R=0.1 と変わらない。
- 振れ幅は最大 25 でさらに増加。
- 緑は最大 6 ノードをもつ集団になる場合もあるが、多くが他の集団に吸収されていく点は R = 0.1 と同じである。

さらに、上記の特徴は、普遍な性質を持つものであるかを R = 2.0, 3.0, 4.0, 5.0 と単調に増加させて観測する。

計算結果は図 4-33 の(a)～(d)までで

- 青と赤について、R = 3.0 以上では青が赤よりも優位。
- ３集団とも観測期間内で消滅せずに残るのは、緑が増加した時で R=0.1 と変わらない。
- 振れ幅は R の増加に伴って急増。緑は R = 5.0 以上では、もはや、他のパーティーもしのぎ、パーティー(０)よりも優位になる場合がある。

第３集団としてパーティー(２)は、１ノードに過ぎなかったが、影響度 R の範囲が大きくなると、同意見の集団が形成され、キャスティングボートを握る集団から寝返りを他の集団に起こす影響力をもつ集団に成長した。さらに、他の集団と規模的には同じ程度にまで影響力を持つようになることがわかる。



(a) R = 2.0　(b) R = 3.0

(c) R = 4.0　(d) R = 5.0

図 4-33　コンテンツの信憑性：100%、同じパーティーの信頼性：100%。10 回計算

４.７　寝返りとしっぺ返し

ここまでの結果を見ると、2 つのメッセージ集団に１集団が入ることで、弱肉強食現象、つまり集団の構成員数（SNS でいえば会員数）が多いほど、他の集団を取り込む力が強い現象が観測できる。さらに図 ４-26 のように同調としっぺ返しを繰り返す現象が起こっている。図 ４-32 のように、第２集団が一時的に残る場合と図 ４-33 のように最後まで第 3 集団として存在する場合もある。

ノードを寝返らすほど接近している（影響度 R の範囲が小さい）と寝返りをお互い行い、相殺されてしまう「しっぺ返し」現象もみられる。

寝返りの寝返りである「しっぺ返し」が起こると、図 ４-26 のように優位である集団が入れ替わる「振動」現象が発生した。しかし、図 ４-29 のように影響度 R の範囲が小さい場合、優位性の振動自体も小さく、最終的にはどちらかの集団に吸収されていく。R が大きいと影響範囲が大きいため、吸収が進まず、優位性の振動によって相殺されないノードが残る。残ったノードが他の集団を寝返らすことで残存する。残存したノードがまた他の集団のノードを寝返らせる。

「しっぺ返し」現象は見方を変えると、寝返りをさらに寝返らせるには自分の R と相手の R が重なる頻度が多いとも言える。例えば、集団 A のノード(a)と集団 B のノード(b)のそれぞれの R が重なることを考えてみる。実際の SNS であれば、ノード(a) は A さんで、ノード(b) は B さんであるとすると、A さんが投稿したメッセージは、集団 A の意見に沿い、人間関係も集団 A の会員と A さんはリンクがあって信頼している状態である。そこに B さんは投稿した A さんのメッセージを見ることができたとする。この時点で B さんは、A さんからの影響を受けるかもしれない。つまり、A さんの影響度 R の範囲に B さんが立ち入ったことと等価である。集団 B に属する B さんであるのでモチベーションは、A さんの意見に反対であると考えられる。これをシミュレーションの処理では、A さんのメッセージは B さんの反対意見を覆し、寝返らせる。つまり、A さんに同調することになる。[41]

気付かれたと思うが、ある前提条件がシミュレーションの処理にある。前提条件は、A さんの意見に同調させることで、B さんは、所属していた集団 B の会員とは反対のメッセージを投稿するだろう。つまり、この前提条件が要因となり、モチベーションは、寝返りやしっぺ返しをメッセージに反映させようとする。

それでは、この前提条件は実際の SNS では何に相当するのであろうか。

それは「暗黙の了解」の規定であると思われる。集団 A の会員が集団 B の会員のメッセージを見た時の反応や行動（今回はメッセージング）で、会員間の「暗黙の了解」で規定されたものである。シミュレーションでは、B さんは集団 B の暗黙の了解の規定として「寝返らせる」ことで、「集団 A の投稿を見たら直ぐに反対の意見をコメントする」ことである。B さんが A さんを SNS で未知の相手である間は、メッセージ空間では影響度 R の範囲の範囲外であることを意味する。A さんと B さんのリンクも存在しないので、寝返りも起こらない。B さんが A さんの投稿を見た時、メッセージ空間では B さんは A さんの R 内に侵入したことになる。このとき、A さんのメッセージの扱い方次第で、B さんは「個人的には同意したいが、所属する B 集団の暗黙の規定で、反対するべきだろう」と考えることになる。

３.２の 1 つのノードで自らのモチベーションの判断でメッセージングを行う際には、メッセージングで生まれてくる集団の形成は考慮していない。SNS の社会性を考えると、ネット上での交流が仲間をつくり、意思疎通をすることは、自然な行為である。交流によって仲間やファンが生まれてくることは、メッセージングによっ

て仲間やファンという別のメッセージの発信源をつくることになる。メッセージングが再帰的にメッセージを作ることになる。

自分の出したメッセージを受けてできる集団からメッセージをさらに受けて、同調や反対で自分のメッセージを変え、相手に同調や反対のコメントをする。シミュレーションで設定した前提条件は、同調ならそのまま、反対なら反対のコメントをして寝返らせるという「暗黙の了解」の規定であると説明できる。

個人のモチベーションだけの段階から集団の規定によって影響を受ける段階に視点が広がったことになる。ヒューマン・コミュケーションは、個人のミクロ的なモチベーションだけでなく、暗黙の規定でできるマクロ的な集団のモチベーションや戦略も考えねばならないことを示している。

4.8　インフルエンサーの導入

これまでの第３勢力は影響度 R や信頼度レベルが極端に大きなものではなく、対峙する集団と同程度であった。実際の SNS では、他のノードに大きな影響を与えるインフルエンサーと呼ばれる会員もいる。インフルエンサーの動きをみるために 50 ノード対 50 ノードの対峙する勢力に１インフルエンサー（緑）が第３勢力として侵入した場合を図 4-34 に示す。

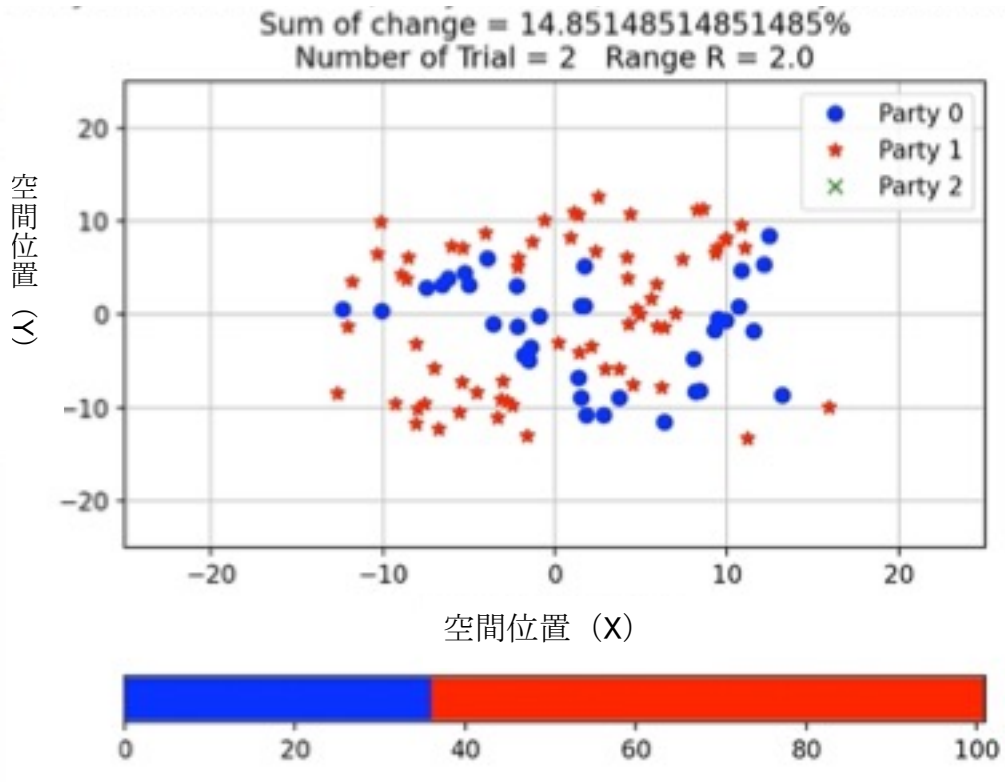


図 4-34　50:50 で１インフルエンサー（緑）が第３勢力　R=2（付録の動画から抜粋）

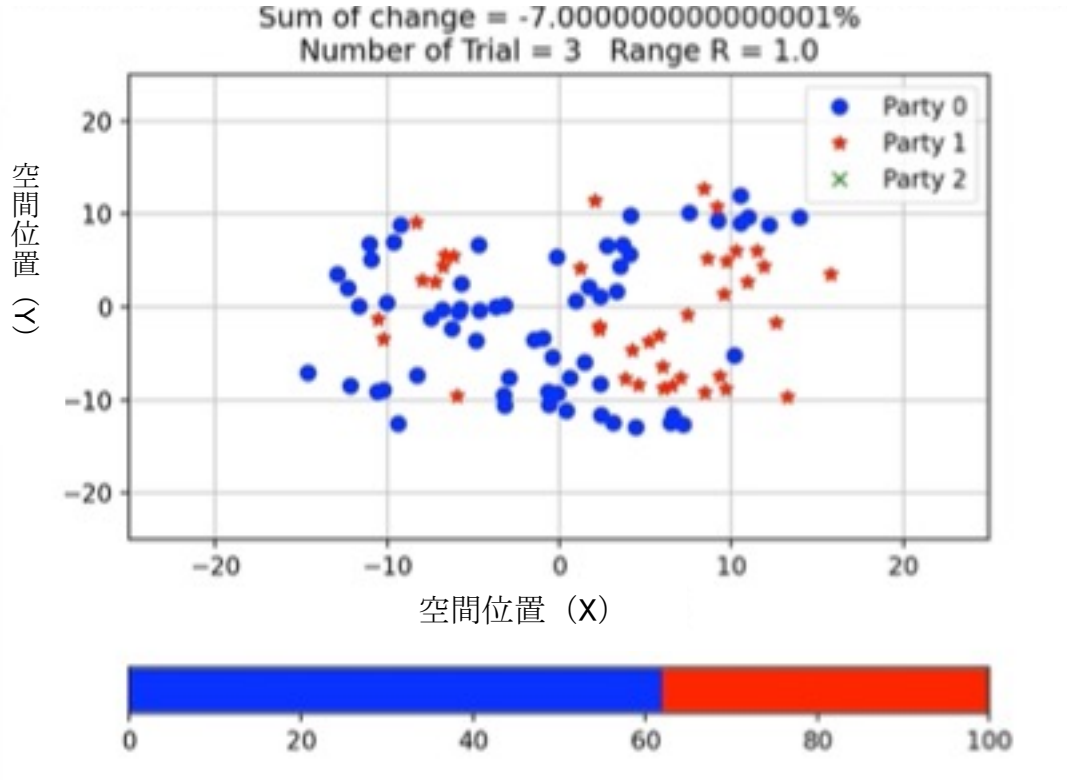


図　4-35　45:45 で 10 のインフルエンサー（緑）が第３勢力　R=1（計算結果の動画から抜粋

45 ノード対 45 ノードの対峙する勢力に 10 のインフルエンサー（緑）が第３勢力として侵入した場合を図 ４-35 図 ４-34　50:50 で１インフルエンサー（緑）が第３勢力　R=2（付録の動画から抜粋）に示す。

結果として、インフルエンサーの影響も対峙するそれぞれ集団の影響度 R よりも大きい場合には集団に与える影響は大きいことがわかった。

４.９　メッセージングによる集団戦略

集団同士のメッセージや意見については、石井らの研究[32]が詳しいので、ここでは、メッセージングにより４.４で述べた集団のメッセージ空間を作る過程に注目する。

2 つの集団の間では、SNS の領域だけでなく一般的な集団行動で論じられているが、本研究のヒューマン・コミュニケーション・ダイナミクスの視点で以下のように分類した。

① 同調／賛同：２集団間でメッセージに対して同調や賛同をそれぞれの集団の構成員（会員）で共有している。同調／賛同するだけではなく、自らの意見をコメントする場合もある。
② 反抗／反対：２集団間でメッセージに対してそれぞれの集団の構成員（会員）が相手のメッセージに反抗あるいは反対することで共有している。
③ 追従：２集団間でメッセージに対してそれぞれの集団の構成員（会員）で相手のメッセージに追従する。①の同調／賛同とは異なり、自らの意見を明らかにしない場合もある。

④ 無視：２集団間でメッセージに対してそれぞれの集団の構成員（会員）があえてコメントや意見を公表せず、両集団間に深い関係がないことを暗黙に示す。

この 2 集団の関係に第３集団が介入していく。

すでに４.５で示したシミュレーションでは②の反抗／反対の状況で相手を寝返り介入する１例を示した。つまり、４.７で述べたシミュレーションのトリックは、この集団戦略の１つである。１ノードから第３集団になることは結果の通りである。

②以外の①、③と④の集団戦略と個人のモチベーションによるメッセージングが、集団の形成にどう関わるかは、今後の展開で考えたい。

４.１０　　オピニオン・ダイナミクスとの関係

ここまでメッセージでの集団形成と集団戦略を論じてきたが、ネットワークでの集団を扱う石井らのオピニオン・ダイナミクスとの関係も示しておきたい。[32]

本研究とオピニオン・ダイナミクスの共通性は、

① 集団の背景にある意見や戦略などのダイナミクスを探り、社会ネットワークに応用すること
② インターネットのような情報基盤自体を論じるものではないこと。情報の「情」に重点をおくこと
③ 本研究のメッセージングと同様に、個々の意見はオピニオン・ダイナミクスにおいても多数の人の影響を受けて意見が形成されていくこと

にある。

一方、本研究とオピニオン・ダイナミクスの理論との違いは、

④ オピニオン・ダイナミクスは社会での合意形成プロセスなどマクロなダイナミクスに注目するが、本研究では、合意形成前のミクロ的な個人のモチベーションに注目すること
⑤ 本研究の焦点がメッセージや意見という結果を導くモチベーションを探ること
⑥ オピニオン・ダイナミクスは投票や SNS の投稿データなどの観測可能なソーシャル・データによる検証が期待できるが、本研究の個人のモチベーションを観測してデータ化するための方法については現在未解決であること

である。

オピニオン・ダイナミクスとの共通性から、本研究は、④から⑥の違いはあるもののオピニオン・ダイナミクスの領域で、SNS に特化したモチベーションとメッセージングを取り扱う一分野と捉えることもできる。

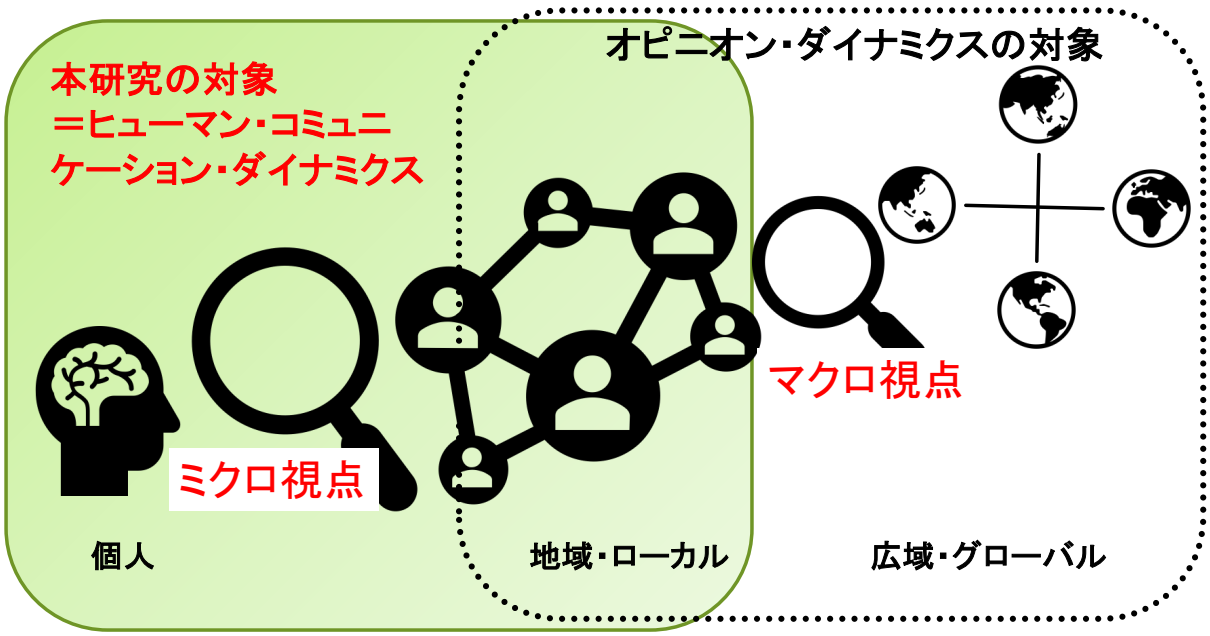


図 4-36 ヒューマン・コミュニケーション・ダイナミクスとオピニオン・ダイナミクスとの関係

図 4-36 は、本研究の対象であるヒューマン・コミュニケーション・ダイナミクスとオピニオン・ダイナミクスの関係を示した図である。図の右にあるようにこれまで情報ネットワーク理論は、地域やローカルコミュニティーから広域でグローバルな地球規模でのマクロな視点で人間の集団的行動や情報の展開などを扱ってきた。これらの構成単位である個人からローカルな集団までのミクロな視点でのモチベーションや意志による動態を扱うのが、ヒューマン・コミュニケーション・ダイナミクスであることを示している。

4.11 ミクロ視点からマクロ視点へ

ミクロ視点でみると、SNS 会員のメッセージに対する信憑性や人間関係の信頼性によって他の会員と集団を形成することを前章で示した。また、形成されたメッセージ集団は、会員のモチベーションに影響を与え、発信するメッセージも影響を受けることを示した。

影響力の大きい会員は、メッセージ空間では多くの会員のメッセージを同調させる力がある。また、2 つのメッセージ集団間に第 3 集団が接触した場合、影響力の大きさ、各集団の会員数で、SNS で起こる幾つかのパターンが発生することが認められた。能動的ノードによる会員のメッセージによる影響を説明するモデルを発展させて、送信されたメッセージによって形成される集団（メッセージ集団）の動態を観察し、SNS と同様な集団間の力学が確認できた。

第５章　今後の展開

ヒューマン・コミュニケーション・ダイナミクス理論を提唱するためにはまだまだ検証しなければならない仮説があり、以下のような議論をしなければならない。また、いくつかの課題が今後の展開として残っている。

５.１　モチベーションと意思、メッセージングの起源

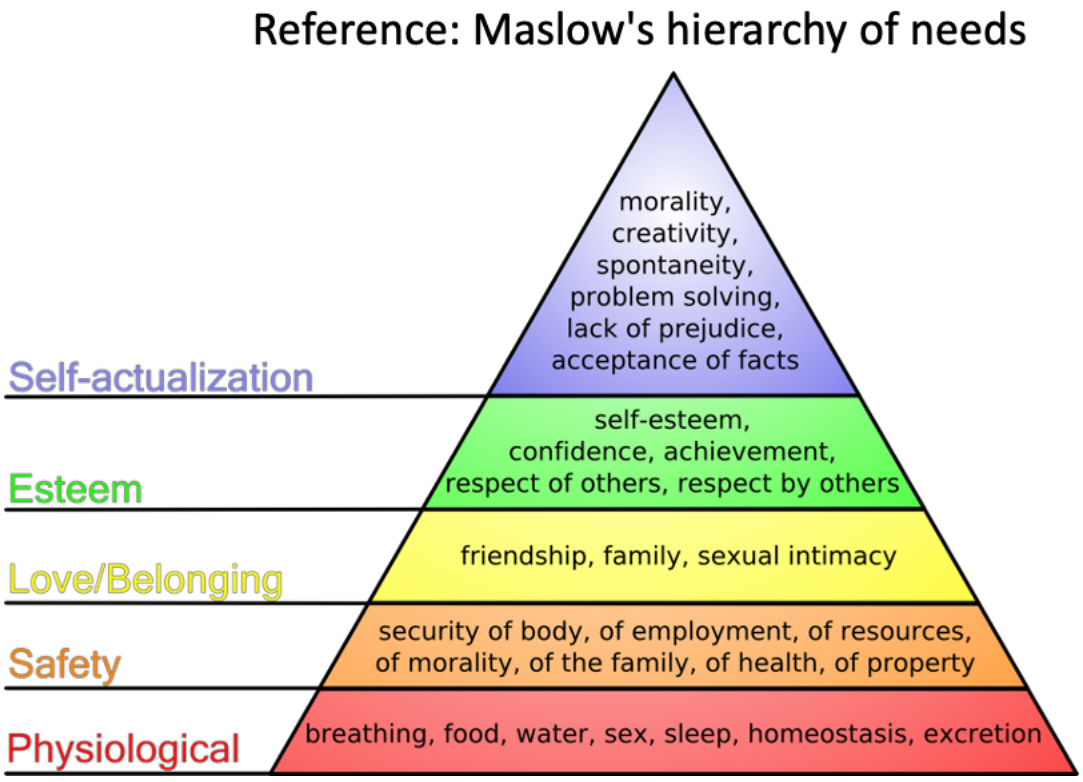


図　5-1　マズローの５段階欲求仮説([39]を抜粋)

SNS は、１.４で述べたように歴史は浅く、人類が世界的に手に入れた最初のコミュニケーションツールであり、見知らぬ相手とでも交流できるツールである。電話や電子メールのように知人とのコミュニケーションで事実や状況を伝えると言った情報の「報」に重きを置いたものとは異なる。言い換えれば、SNS は情報の「情」に密接に関わり、第３章で導入したモデルでその性質は観測できるものの、根源的なモチベーションの何たるかについては推論の域を出ていない。

図　5-1 に示すマズローの５段階欲求仮説[39]にしても、欲求とモチベーションの結びつきは直接示されてはいない。ただ、この仮説で示されるのは、人間の欲求には段階があり、ある欲求が満たされると上位の欲求を満たすようになるという点である。この仮説にあえて従うと、電話や電子メールの時代には、下から３番目の社会帰属や愛情（社会的欲求）の欲求までが限界であったのかもしれない。４番目の尊厳（自我や自信）、五番目の自己実現の段階は、これまではインターネットには無関係で個人がその欲求を充足してきた。

SNS の登場は、インターネット上にヒューマン・コミュニケーションを構築する能力を持たせ、物理的に孤立していても、インターネット環境さえあれば、世界の

会員同士が、自分の意見や主張、悩みといった人間独特の 4 番目や 5 番目までも充足させることができるようになったといえる。[16][17][39]

数年前からマーケティング業界で話題になってきた YouTuber（ユーチューバー）や influencer（インフルエンサー）はインターネット上の SNS でのポジション（位置付け）である。インフルエンサーは必ずしも芸能人や有名人とは限らず、狭い特定の領域（niche）で専門家であり、オピニオン・リーダーで SNS での影響力が大きい（フォローしている会員が極端に多いのが特徴）会員である。ちょうど 4.8 で述べた信頼度レベルが高く、影響範囲が大きい会員である。[17]

インフルエンサーは、マズローの 5 段階欲求仮説で言えば、第 5 段階の欲求を満たす人たちである。マーケティング業界で話題になったのは、特定の商品をインフルエンサーが取り上げてくれれば、インフルエンサーの影響で同調者の購買意欲を上げることが期待できるからである。[16][17] [40][39]

これまで導入してきたイベント・ドリブン・モデルで 1 ノードから拡散するメッセージングも、モチベーションの視点をマズローの欲求仮説の欲求や意思の実行レベルとみると、図 5-2 のように、ノードでの判断の背景に欲求による要請でモチベーションが起こり、メッセージを起こすといった仮説も設定できる。

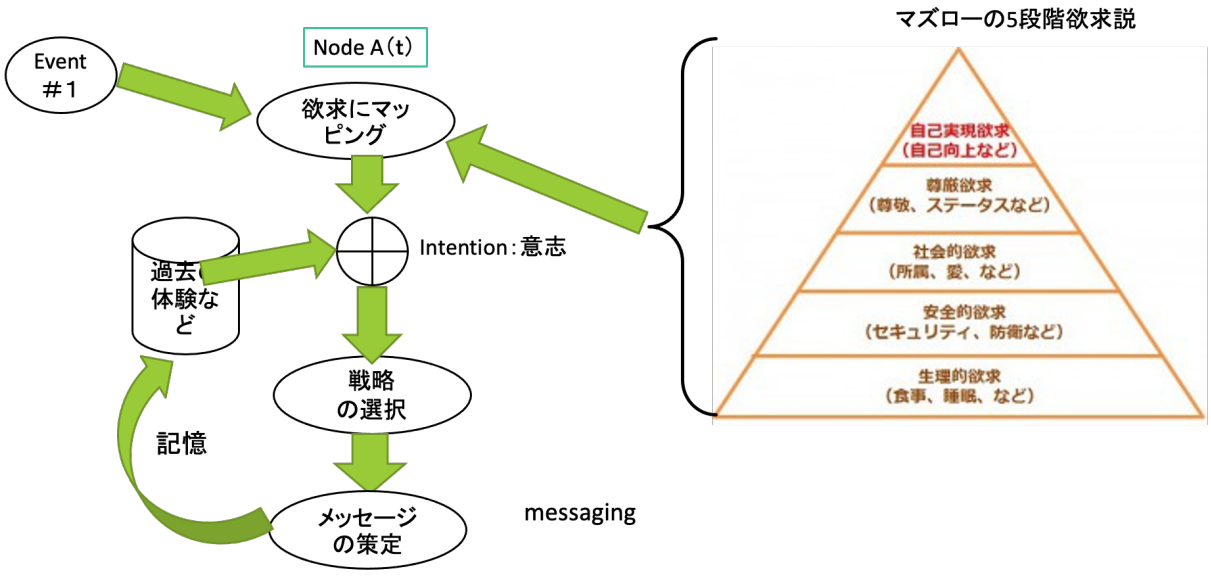


図 5-2 マズローの 5 段階欲求仮説と導入したモデル

SNS でこの仮説を限定的に適応させることで検証ができる可能性がある。例えば、実験的に被験者とその関係者のみで構成された閉鎖的な SNS を作り、被験者に感情の高ぶりを抑えられない状況で、SNS にメッセージを投稿してもらう。関係者はそれに対するコメントを行う。実験後、メッセージ 1 つ 1 つに関連する欲求をアンケートで特定する。この実験で図 5-2 の欲求へのマッピングをモチベーションと捉えれば、このモデルが検証できる可能性がある。

5.2 メッセージ集団の展開

第 4 章で述べたメッセージ集団とモチベーションについて更なる展開が必要と思われる：

① 論文では、影響度 R の範囲にあるとき、メッセージの受信者はメッセージに同調したが、反発や無視といった別の判断も存在する。このような場合想定されるメッセージ集団の動態はどうか

寝返りについて述べたが、相手に対して４.９のような組み合わせが考えられる。例えば、無視のような場合は、単純にメッセージを無視するだけでなく、無視することで、無関係状態を作るといった戦略も存在する。このような戦略の記述を検討しなければならない。

② 上記の判断は各ノードで一律に行われるが、これをランダムに行ったときはどのような動態になるか。インフルエンサーの影響はどうか。

集団も会員と同様、戦略がいつも同じであるということはない。集団間で迎合したり、反発したりが存在する。このような状態で、会員の離反などの現象が起こる可能性もある。

インフルエンサーの集団などが想定される場合マーケティングでの仮説が通用するか検証する。

③ メッセージ集団を安定あるいは残存させるために条件は何か

SNS のメッセージによる集団がどのような状況で安定的に存在あるいは残存するのかを見出すもので、小売業などのマーケティングでは重要な情報を見出せる可能性がある。

④ メッセージ集団の意図的な戦略（集団戦略）を実行する方法は何か

シミュレーションでは、外部から設定で可能だが、これをイベントやメッセージで制御できるかについて、信憑性や信頼性、メッセージの質やタイミングなどから集団の暗黙の規則に変える手段を特定できねばならない。

⑤ オピニオン・ダイナミクス理論との関係で、メッセージ集団戦略で同じ結果が導けるのではないか

集団のメッセージ戦略を合意形成というゴールで整理して、信頼度レベルや影響度 R の範囲を考えると、石井らの Trust and Distrust model の結果を裏付ける結果になる可能性もある。[32]

⑥ 集団戦略を設定した上でのメッセージの拡散制御の方法

これは、③および④の仮説で条件が見出せれば可能である。これがわかれば産業界への応用ができる。

⑦ 限定的な SNS 環境での社会実験で本理論を検証する。

５.１にもあるように、実際の SNS で行うにはまだまだ検証が足らないが、限定的な実験環境でメッセージの拡散や集団戦略に対するゲームを設定して段階的に検証する方法など探索する必要がある。

⑧ 初期位置（陣形）による集団行動の影響をみる。

第４章では 2 次元のランダムウォークでモデルを構成したが、図 ４-30 のように初期位置によって第３勢力の生き残り状態が変化する場合も想定できる。上記の③および④の解決を与える可能性もある。

⑨ 時限的に影響度 R が変化する

影響度 R に着目し、帰属集団の影響を時限装置のように一定の時間になれば寝返るといった集団心理を入れて動態を分析する。

まとめ

本研究は、自分が投稿したメッセージを会員の誰が見ているのかさえわからない SNS で、正確に相手に意図が伝わっているかどうかも気にせず、他の会員との交流が続けられるのかという疑問が発端であった。本論文は、インターネットという情報の「報」の力を使って、メッセージという「情」を伝えようとするモチベーションにこの疑問を解く鍵があると考えた。理論的にこのモチベーションを数理モデルのパラメータに置き換え、その動態を分析することで、これまで捉えどころなかったモチベーションの働きや行動に対する影響を明らかにした。

第１章では、SNS が普及する以前からあるヒューマン・コミュニケーションを省みることで、メッセージを発信するモチベーションにコミュニケーションの媒体が必ずしもインターネットである必要はないことについて述べた。また、メッセージを投稿（送信）するモチベーションは、投稿に際して手に入れた外部の情報やコンテンツの信憑性と投稿者の人間関係の信頼性に依存していると仮定している。なお、本論文での SNS は Twitter や Facebook といった特定のサービスを指すのではなく、これらに共通した機能に絞って論証している。

第２章は、本研究の位置付けを明確にするために、SNS でメッセージを投稿する行為（メッセージング）を情報科学分野と認知科学分野の両面から調査した。情報の「情」の研究では認知科学を、「報」の研究では情報科学をそれぞれ調査した。モチベーションを理論的に記述することや最適なモデルを構築するためにも、先行研究で使われているモデルの特質を抽出した。ヒューマン・コミュニケーションの動態をモデル化する際にこれらの特質を利用できないかを検討した。モチベーションの特徴を記述できる拡散方程式の数理モデルをまず導入した。これにモチベーションからの行動（メッセージング）が記述できるイベント・ドリブン・モデルを組み合わせ、第３章のシミュレーションで利用した。

第３章ではメッセージの送受信が行われるインターネットの仮想空間内でメッセージ空間を定義した。メッセージ空間では会員はノードとみなし、イベントのお知らせやメッセージの投稿が閲覧できる状態をリンクに置き換えた。メッセージの内容（コンテンツ）の信憑性や会員の人間関係による信頼性を変数とした。また、コンテンツは会員のモチベーションによって改変や編集が行われるので改変量を変数とした。これらの変数を用いて導入した数理モデルを計算し、シミュレーションした。

外部からのイベント情報をきっかけに、１つのノード、つまり会員一人がメッセージを投稿する状況を用いて、シミュレーターの検証を実施した。結果として想定したメッセージの拡散が認められ、シミュレーションの正当性が確認できた。次に実際の SNS の会員のように、シミュレーターのノードでパーソナライズを設定した。パーソナライズとは、各ノードに受信したメッセージに人間が投稿する際に行う判断を設定するものである。

パーソナライズ（レベル１）では、処理の判断をコンテンツの編集の度合いや信憑性、会員の信頼性をランダムに設定してメッセージの拡散を観測した。コンテンツの編集の度合いや信憑性、会員の信頼性がメッセージングに大きな影響を与えた。送信者の意図的なコンテンツの編集は拡散せず、また信憑性や信頼性が低いものは拡散しない結果となった。

パーソナライズ（レベル２）では、各ノードのしきい値をランダムに設定し、レベル１と同様に拡散を確認した。結果から、しきい値は、メッセージの拡散を抑えるブレーキの役割を果たしていることがわかった。
パーソナライズ（レベル３）では、レベル１とレベル２のランダム化を同時に設定し、より実際の SNS の状態に近づけた。メッセージの拡散に影響を与え、SNS のように会員がゲートキーパー（門番人）のように極端なメッセージを拡散しないケースを観測した。
パーソナライズ（レベル４）では、メッセージに肯定・否定の極性を条件に追加した。メッセージが同じ極性（肯定を正、否定を負とした）であればメッセージの拡散は進むが、極性が異なると、メッセージが拡散しないことが結果として得られた。
パーソナライズ（レベル５）では、メッセージを投稿したという経験を記憶し、モチベーションのしきい値を記憶に応じて変動させた。数回のイベントからノードにコンテンツを拡散し、しきい値の変動の影響を観測したところ、しきい値をランダムに変動させる状況と同様に、記憶（履歴）によるメッセージの拡散への影響が大きい結果を得た。さらに記憶したしきい値を嵩上げすると、一様に拡散にブレーキが掛かった。記憶を繰り返し呼び起こす恣意的な広告宣伝などによる嵩上げ現象はデジタル・マーケティング戦略に応用できるかもしれない。メッセージを拡散する場合にコンテンツの信憑性、人間関係の信頼度を過去の履歴も参考にすることで、不用意な拡散を抑制する効果があることが分かった。
社会現象として buzz（話題になること）を過剰に起こさせないためにも、コンテンツの検証、信頼関係のある人間関係の確証、さらに過去の投稿の検証が重要であることがこれらの結果から分かった。 逆に、意外にも口コミのような自然発生的に拡散することは頻度が高くないことがわかった。
各ノードのモチベーションによる判断を組み込んだイベント・ドリブン・モデルを使って SNS で起こっている１ノードからのメッセージの拡散現象を再現した。
第４章では、一人の会員からイベントの情報を拡散するのではなく、会員が同時に他の会員やイベントのメッセージを受けた場合、メッセージを会員で共有しながら展開する現象をモデル化することを考えた。また、従来はインターネットの通信機能と SNS で扱うメッセージを分けて考察してきた。ここではヒューマン・コミュニケーションの個人から集団までの一連の動態に着目するためにメッセージの伝搬に重点を置くモデルを導入した。第３章のイベント・ドリブン・モデルでは、各ノードでの自己判断は初期設定で一括して行われる受動的ノードであった。これでは、他のノードからの影響をダイナミックに記述できない。各ノードがメッセージを受け取った後も独立してメッセージの処理を行える能動的ノードに変える必要が出てきた。
能動的ノードに変えたエージェントモデルで各ノードと集団形成、集団からの会員のモチベーションの影響を記述した。リンクの強度（親密度）に反比例してノード間の距離を決めた。つまり親密度が高いと両ノードは接近している状態となる。発信者は、メッセージの信憑性と発信者との人間関係での信頼性でメッセージをさらに送信するか、編集して送信するか、送信しないといった判断を行う。さらに、メッセージの信憑性と発信者との人間関係での信頼性の積を信頼度レベルとしてメッセージの到達距離に比例させ、信頼度レベルが高いとメッセージも遠方にまで到達することとした。さらに、会員のリンクは他の会員への影響を与える経路であるこ

とから、リンク数を影響度 R の範囲として、エージェントモデルに組み込んだ。SNS のメッセージの交流とリンクの構造を 2 次元平面で信頼度レベルに応じてランダムウォークするエージェント（ノード）を設定した。影響を及ぼす範囲 R を各ノードに持たせ、R 内で接触することでリンクが生じるモデルとした。信頼度レベルが声の大きさで、影響度 R の範囲が声の届く範囲、接触がリンクの確立としたイメージのモデルである。

シミュレーションでは、メッセージの信憑性、人間関係の信頼性、帰属する集団の属性を持たせ、信憑性と信頼性の積である信頼性レベルと影響度 R の範囲を変えることで、個人から集団へのダイナミクスを観測することにした。シミュレーターを最初に自明なパラメータで検証後、同一意見を持つ集団に 1 名が接触する場合、影響度 R の範囲を変化させて動態を観測した。なお、集団からの影響を明確にするために、ノードがお互いに接触した後、集団のノードでも侵入するノードでも相手を寝返らせる戦略を持つとした。

影響度 R が狭い範囲で信頼度レベルを下げると、寝返りが少なくなり、侵入者のメッセージが抑え込まれる結果が観測できた。

さらに信頼度レベルを 100%で一定として影響度 R を変化させた。興味深いことに影響度 R が小さい集団では、自分の所属する集団を拡大することは難しく、一定の R 以上になると、他の集団のノードを寝返らせる方が多くなることが観測できた。これは、一定の影響範囲を持たないと集団が維持できないことを意味している。

次に 2 つの対抗する集団に最初は 1 会員の第３集団が侵入する場合を計算した。同数の集団に１ノードが侵入する場合、侵入者がキャスティングボートを握る現象が観測できた。

R が大きいと２集団のうちどちらかが侵入者を飲み込むか、生き残る現象も観測できた。さらに、生き残りのケースを分析すると、ノードを寝返らすほど接近している（影響度 R の範囲が小さい）場合、寝返りをお互い行い、相殺されてしまう「しっぺ返し」現象もみられた。寝返りの寝返りである「しっぺ返し」では、優位性が集団間で入れ替わる振動が発生した。影響度 R の範囲が小さい場合、優位性の振動自体も小さく、最終的にはどちらかの集団に吸収されていった。影響度 R の範囲が大きいときは、吸収が進まず、優位性の振動によって相殺されないノードが残った。残ったノードが他の集団を寝返らすことで残存したことを示した。これは SNS でも対立する意見は、局所的で小さな範囲でのメッセージであればどちらかが優勢で第 3 の意見はかき消される。声が大きい（メッセージの届く範囲が広い）場合、対立意見はそのまま残り、第 3 の意見も場合によっては残ることを示した。

また、影響度 R が大きく、信頼度レベルも高いインフルエンサーが第３勢力になる場合も計算した。興味深いことに第 3 勢力がお互いに対峙する集団よりも影響度 R が大きく、信頼度レベルも高い場合、第３勢力は生き残り、他の集団に影響を与えることが観測できた。

最後に本研究とオピニオン・ダイナミクスとの関係について述べた。最近、社会物理学のみならず、政治学や経済学でも注目されているオピニオン・ダイナミクスと本研究には多くの共通性がある、本研究は、多少の相違はあるもののオピニオン・ダイナミクスの領域で、SNS に特化したモチベーションとメッセージングを取り扱う一分野と捉えることができる。また、今後の展開として、第 5 章に示したように行動心理論や認知理論で対象とする人間のモチベーションと本研究のメッセー

ジングのメカニズムについてさらに考察したい。第４章で用いた「寝返り」戦略だけでなく他の集団戦略への展開も考えていきたい。
以上から本論文の目的であるメッセージを投稿するモチベーションが、コンテンツの信憑性と投稿者の人間関係の信頼性に依存していることや属する集団の影響を受けることを確認できた。

個人のミクロ的なモチベーションは、マクロな社会システムに比べ一見小さく無関係に思える。しかし、2020 年初頭から世界中に蔓延した新型コロナウイルスの感染拡大と同様に、微小なウイルスが社会ネットワークを通じてパンデミックを起こしたことも事実である。本研究によって、第３章、第４章で考察したように人間自らの意思（モチベーション）で SNS 上のメッセージを意図的に社会的規模に広げることや、逆に抑止できる可能性が見いだせた。今後、本研究から人間がこの可能性を確実なものにできれば、メッセージングの制御や応用の道が拓けるという希望が持てた。

謝辞

私事とは申せ、長期の休学を余儀なくされたことで研究計画を大きく変えた状況となりました。石井晃教授におかれましては、私のこの状況を深くご理解していただき、さらに研究に対して暖かいご指導をいただきました。おかげさまで私がここまで本研究を進めることができたのも、ひとえに主指導にあたっていただいた石井晃教授があってのことであり、こころよりお礼申し上げます。
本論文の予備審査直後に石井晃先生が急逝されましたため、直接お礼が申し上げられなくなったことは誠に残念でなりません。

古川勝先生にはおかれましては、主指導をその後受け持っていただき、適切な論文指導を始め全面的なご支援をいただき、感謝に堪えません。

鳥取大学工学部の教員ならび教務課のみなさま、石井研究室の大学院生、大学生のみなさまには研究の継続や示唆をいただき、深くお礼を申しあげます。

また、私が代表取締役を拝命しています NVD 株式会社の役員各位と従業員には会社経営の中で研究という機会を与えていただいたことを深く感謝をしております。
私が別の専門分野で教鞭を取っておりますデジタルハリウッド大学学長　杉山知之教授におかれましては、私の鳥取大学大学院へ入学についてもご理解していただいた上に、推薦状までも賜り、深くお礼を申し上げます。

デジタルハリウッド大学大学院事務局と同松本研究室のみなさまにも多忙な中、個人的な研究の時間を与えていただき、ご協力をいただいたことを深くお礼を申し上げます。

最後に働きながら博士課程進学を快諾してくれ、その後の研究期間には精神的支えとなった妻　順子に心から感謝します。

松本　英博

引用文献

[1] 総務省情報通信政策研究所, 情報通信メディアの利用時間と情報行動に関する調査」の「図表 1-1-1-11　代表的 SNS の利用率の推移（全体）」

[2] モバイルインターネット普及率（出典：PwC「グローバルエンタテイメント＆メディアアウトルック 2018-2022」

[3] JPNIC アーカイブス, インターネット歴史年表, https://www.nic.ad.jp/timeline/

[4] 増田直己, 今野紀雄, 複雑ネットワークの科学, 産業図書, pp. 33-158, ISBN-978-4-7828-5151-7 (2005)

[5] 山口真一, ネット炎上の実態と政策的対応の考察- 実証分析から見る社会的影響と名誉毀損罪・制限的本人確認制度・インターネットリテラシー教育の在り方―, 総務省 情報通信政策レビュー 第 11 号, pp.52-74 (2015)

[6] Franklin, Stan, Artificial Minds Boston, MA: MIT Press (2005)

[7] 大向 一輝, SNS の現在と展望--コミュニケーションツールから情報流通の基盤へ, 情報処理 : 情報処理学会誌 : IPSJ magazine, pp. 993～1000 (2006)

[8] 鳥海不二夫 編著, 計算社会科学入門, 発行元,丸善出版, ISBN 978-4-621-30596-6 (2021)

[9] 藤原正弘, 木村忠正, インターネット利用行動と一般的信頼・不確実性回避との関係, 日本社会情報学会学会誌 20(2), 43-55, 2009-03

[10]山本晶, キーパーソン・マーケティング: なぜ、あの人のクチコミは影響力があるのか, 東洋経済新報社, ISBN4492557423, 9784492557426 (2014)

[11]森泰親, 大学講義シリーズ　制御工学, コロナ社, ISBN978-4-339-00143-3 (2001)

[12]久保川達也著,新井 仁之, 小林 俊行, 斎藤 毅, 吉田 朋広, 現代数理統計学の基礎, 共立出版, ISBN　978-4-320-11166-0 (2017)

[13]岸野正剛, 量子力学―基礎と物性, 裳華房, ISBN-10 : 4785320729 (1997)

[14]近藤次郎, 数学モデル 現象の数式化, 丸善, ISBN 3041-2123-7924 (1976)

[15]長岡洋介, 電磁気学 I 電場と磁場 (物理入門コース 新装版), 岩波書店, ISBN-10-4000298631(2017)

[16] 松本英博, 図解入門ビジネス 最新 事業計画書の読み方と書き方がよ～くわかる本[第 3 版], 秀和システム, ISBN-10 ： 4798044490 (2015)

[17]本田哲也, 池田紀行, ソーシャルインフルエンス 戦略 PR×ソーシャルメディアの設計図, アスキー, ISBN-10 4048864432: (2012/6/11)

[18]Wikipedia の参照：「コミュニケーション」を検索：https://ja.wikipedia.org/wiki/%E3%82%B3%E3%83%9F%E3%83%A5%E3%83%8B%E3%82%B1%E3%83%BC%E3%82%B7%E3%83%A7%E3%83%B3

[19]Wikipedia の参照：「通信」を検索：https://ja.wikipedia.org/wiki/%E9%80%9A%E4%BF%A1

[20]Barry Wellman, Computer Networks: As Social Networks, SCIENCE, 14 Sep 2001, Vol 293, Issue 5537, pp. 2031-2034 DOI: 10.1126/science.1065547

[21]Lazer D1, Pentland A, Adamic L, Aral S, Barabasi AL, Brewer D, Christakis N, Contractor N, Fowler J, Gutmann M, Jebara T, King G, Macy M, Roy D, Van Alstyne M, Social science. Computational social science, Science, 01 Feb 2009, 323(5915):721-723, DOI: 10.1126/science.1167742 PMID: 19197046 PMCID: PMC2745217

[22]M. A. Nowak, A. Sasaki, et.al, Emergence of cooperation and evolutionary stability in finite populations, Nature 428, pp.646-650 (2004)

[23]Moran, P.A.P. The statistical Processes of Evolutionary Theory, Clarendon Press, Oxford, (1962)

[24]Yoshimi Yoshino, Naoki Masuda, Evolution of cooperation is a robust outcome in the prisoner's dilemma on dynamic networks, 7th Network Symposium 2011 (2011)

[25]M. Tanaka, T. Murakami, Co-evolution of Strategies and Players' Network in the Reputation based Prisoners' Dilemma (RPD), Meeting abstracts of the Physical Society of Japan 64(2-2), 207(2009)

[26]J. M. Pacheco Francisco C. Santos, Max O. Souza, Brian Skyrms, Evolutionary Dynamics of Collective Action, The Mathematics of Darwin's Legacy. pp. 119-138 (2011)

[27]吉田就彦, 石井晃, 新垣久史,大ヒットの方程式 ソーシャルメディアのクチコミ 効果を数式化する, ディスカヴァー・トゥエンティワン社(2010)

[28]David Easley, Jon Kleinberg, "Networks, Crowds, and Markets: Reasoning About a Highly Connected World", Cambridge University Press, pp.563-687 (2010)

[29]湯田聰夫,小野 直亮 , 藤原 義久, ソーシャル・ネットワーキング・サービスにおける人的ネットワークの構造, 情報処理学会論文誌 47(3), 865-874, 2006-03-15

[30]岡田直之, 脳に宿る心 -認知科学・人工知能から神秘の世界に迫る -, オーム社, ISBN-978-4-274-20783-9 (2009)

[31]安西祐一郎, 心と脳 -認知科学入門, 岩波新書, ISBN-10 ： 4004313317 (2011)

[32]Akira Ishii, Ippei Yomura and Nozomi Okano, Opination Dynamics Including both Trust and Distrust in Human Relation fo Various Network Structure, In: Proceedings of TAAI 2020

[33]David Easley, Jon Kleinberg, Networks, Crowds, and Markets: Reasoning About a Highly Connected World, Cambridge University Press, pp.563-687 (2010)

[34]Ishii Akira and N Okano, Sociophysics approach of simulation of mass media effects in society using new opinion. In: Advances in Intelligent Systems and Computing as the Proceedings of IntelliSys2020, K. Arai et al. (Eds.): IntelliSys 2020, AISC 1252, pp. 13- 28, 2021.

[35]Hidehiro Matsumoto and Akira Ishii. 2020. An Analysis Approach of Messaging Mechanism on Social Networking Services. IEEE International Conference on Big Data, DOI: https://doi.org/10.1109/BigData50022.2020.9378013

[36]Hidehiro Matsumoto and Akira Ishii. 2021. Proceedings of SAI Intelligent Systems Conference. IntelliSys 2021: Intelligent Systems and Applications pp 876-893An Analysis with Dynamics Between Human Motivation and Messaging on Social Networking Services. DOI: https://doi.org/978-3-030-82193-7_59, EID: 2-s2.0-85113196197

[37]Wikipedia の参照 ： 「情報空間」 を検索 ： https://ja.wikipedia.org/wiki/%E6%83%85%E5%A0%B1%E7%A9%BA%E9%96%93

[38]A. Ishii, H. Arakaki, N. Matsuda, S. Umemura, T. Urushidani, N. Yamagata, N. Yoshida, The 'hit' phenomenon: a mathematical model of human dynamics interactions as s stochastic process, New Journal of Physics 14 (2012) 063018, pp. 22

[39]小高良知, マズロー欲求階層説と臨床社会学, 東海学院大学紀要,巻 4, pp. 53 - 59 (2011/03/20)

[40]フィリップ・コトラー, コトラーのマーケティング 4.0 スマートフォン時代の究極法則, 朝日新聞出版, ISBN-10 4023316156 (2017/8/21)

[41]R. アクセルロッド, つきあい方の科学: バクテリアから国際関係まで (Minerva21 世紀ライブラリー), ミネルヴァ書房, ISBN-10 ： 4623029239 (1998/5/20)

[42] 合原 一幸 , 神崎 亮平, 理工学系からの脳科学入門, ISBN-10 ： 4130623044 (2008)

[43] 合原一幸, 社会を変える驚きの数学 (ウェッジ選書 32 地球学シリーズ), ウェッジ, **ISBN-10-**4863100256 (2008)

[44]Bing Wang, Lang Cao, Hideyuki Suzuki & Kazuyuki Aihara, Safety-Information-Driven Human Mobility Patterns with Metapopulation Epidemic Dynamics, Scientific Reports volume 2, Article number: 887 (2012)

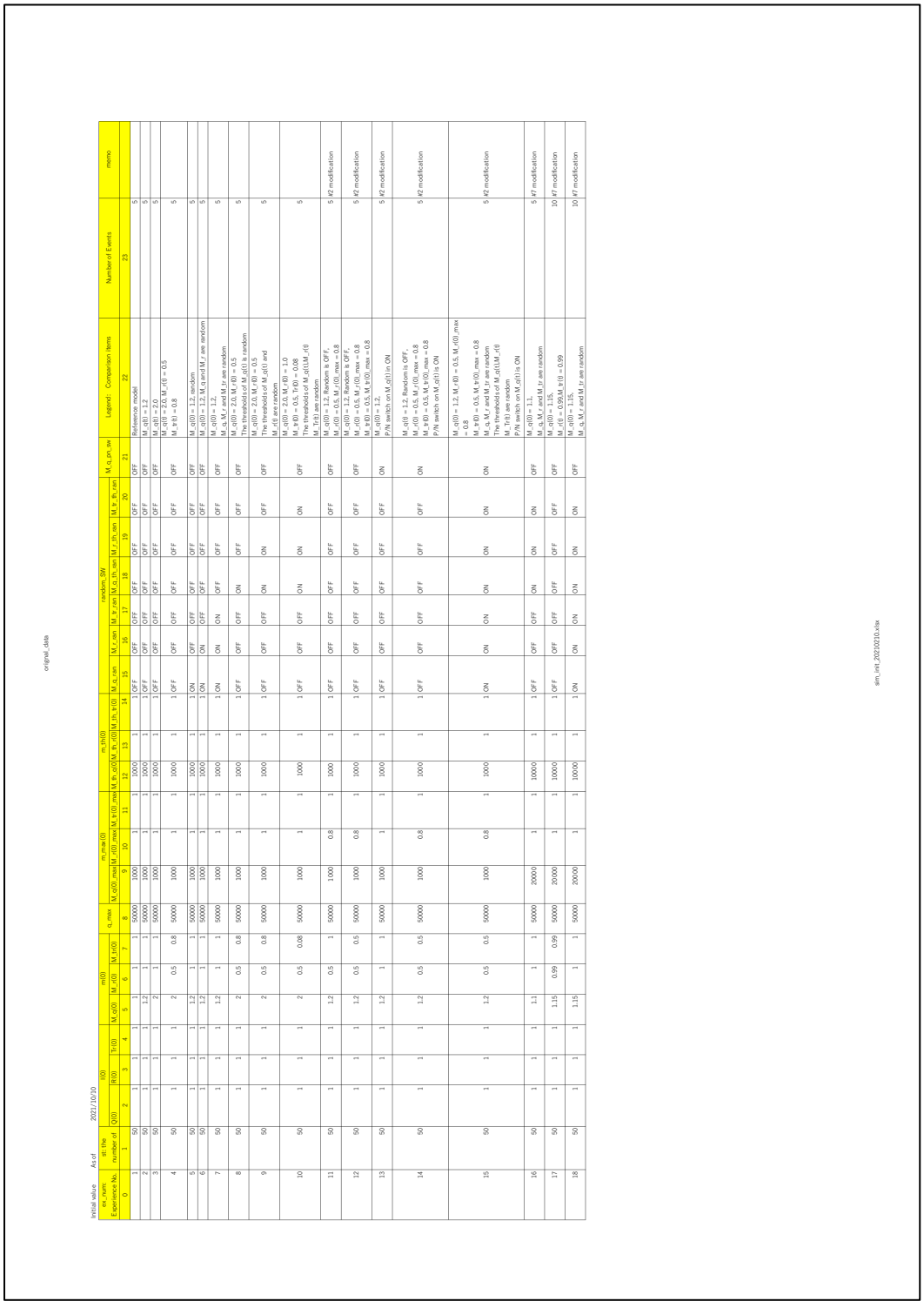
[45]池田泰成, 郷香野子, 馮昕, 庄映琮, 張しん妍, 劉シュウミン, 濱岡豊, インターネット上のオピニオン・リーダーの盛衰, 電子情報通信学会 情報科学技術フォーラム (FIT) (2016/08/23)

付録

本研究で作成したシミュレーションのソースコードや動画の結果を以下に示す。
Python 3.0 以上でコーディング。実行環境は Microsoft Visual Code Studio を推奨

付録１：第 3 章のシミュレーションのソースコード

- シミュレーターの初期化ファイルの例。これを csv ファイルに変換して初期化設定をおこなう。



initial value
As of 2021/10/10
original_data
sim_init_20210210.xlsx

Experience No.	At the number of	Q(0)	R(0)	Tr(0)	M_q(0)	M_r(0)	M_tr(0)	q_max	M_q(0)_max	M_r(0)_max	M_tr(0)_max	M_th_q(0)	M_th_r(0)	M_th_tr(0)	M_q_ran	M_r_ran	M_tr_ran	random_SW M_q_th_ran	M_r_th_ran	M_tr_th_ran	M_q_pn_sw	Legend : Comparison Items	Number of Events	memo
0	1	50	1	1	1	1	1	50000	1000	1	1	1000	1	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	Reference model	23	5
	2	50	1	1	1	1.2	1	50000	1000	1	1	1000	1	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	M_q(0) = 1.2		5
	3	50	1	1	1	2	1	50000	1000	1	1	1000	1	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	M_q(0) = 2.0 M_r(0) = 2.0, M_r(0) = 0.5		5
	4	50	1	1	1	2	0.5	50000	1000	1	1	1000	1	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	M_tr(0) = 0.8	5	
5		50	1	1	1	1.2	1	50000	1000	1	1	1000	1	1	ON	OFF	OFF	OFF	OFF	OFF	OFF	M_q(0) = 1.2, random	5	
6		50	1	1	1	1.2	1	50000	1000	1	1	1000	1	1	ON	ON	OFF	OFF	OFF	OFF	OFF	M_q(0) = 1.2, M_q and M_r are random	5	
7		50	1	1	1	1.2	1	50000	1000	1	1	1000	1	1	ON	ON	ON	OFF	OFF	OFF	OFF	M_q, M_r and M_tr are random	5	
8		50	1	1	1	2	0.5	50000	1000	1	1	1000	1	1	OFF	OFF	OFF	ON	OFF	OFF	OFF	M_q(0) = 2.0, M_r(0) = 0.5 The thresholds of M_q(t) is random	5	
9		50	1	1	1	2	0.5	50000	1000	1	1	1000	1	1	OFF	OFF	OFF	ON	ON	OFF	OFF	M_q(0) = 2.0, M_r(0) = 0.5 The thresholds of M_q(t) and M_r(t) are random	5	
10		50	1	1	1	2	0.5	50000	1000	1	1	1000	1	1	OFF	OFF	OFF	ON	ON	ON	OFF	M_q(0) = 2.0, M_r(0) = 1.0 M_tr(0) = 0.5, Tr(0) = 0.08 The thresholds of M_q(t),M_r(t) M_Tr(t) are random	5	
11		50	1	1	1	1.2	0.5	50000	1000	0.8	1	1000	1	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	M_q(0) = 1.2, Random is OFF, M_r(0) = 0.5, M_tr(0)_max = 0.8	5	#2 modification
12		50	1	1	1	1.2	0.5	50000	1000	0.8	1	1000	1	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	M_q(0) = 1.2, Random is OFF, M_r(0) = 0.5, M_tr(0)_max = 0.8 M_tr(0) = 0.5, M_tr(0)_max = 0.8	5	#2 modification
13		50	1	1	1	1.2	1	50000	1000	1	1	1000	1	1	OFF	OFF	OFF	OFF	OFF	OFF	ON	M_q(0) = 1.2. P/N switch on M_q(t) is ON	5	#2 modification
14		50	1	1	1	1.2	0.5	50000	1000	0.8	1	1000	1	1	OFF	OFF	OFF	OFF	OFF	OFF	ON	M_q(0) = 1.2, M_r(0) = 0.5, M_tr(0)_max = 0.8 M_tr(0) = 0.5, M_r(0)_max = 0.8 M_tr(0) = 0.5, M_r(0)_max = 0.8 P/N switch on M_q(t) is ON	5	#2 modification
15		50	1	1	1	1.2	0.5	50000	1000	0.8	1	1000	1	1	ON	ON	ON	ON	ON	ON	ON	M_q(0) = 1.2, M_r(0) = 0.5, M_tr(0)_max = 0.8 M_tr(0) = 0.5, M_r(0)_max = 0.8 M_q, M_r and M_tr are random The thresholds of M_q(t),M_r(t) M_Tr(t) are random P/N switch on M_q(t) is ON	5	#2 modification
16		50	1	1	1	1.1	1	50000	20000	1	1	10000	1	1	OFF	OFF	OFF	ON	ON	ON	OFF	M_q(0) = 1.1. M_q, M_r and M_tr are random	5	#7 modification
17		50	1	1	1	1.15	0.99	50000	20000	1	1	10000	1	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	M_q(0) = 1.15. M_r(t) = 0.99,M_tr(t) = 0.99	10	#7 modification
18		50	1	1	1	1.15	1	50000	20000	1	1	10000	1	1	ON	ON	ON	ON	ON	ON	OFF	M_q(0) = 1.15. M_q, M_r and M_tr are random	10	#7 modification

- ３.２.３　パーソナライズ（レベル 1）から３.２.７　パーソナライズ（レベル 5）までのシミュレーターのソースコード

```
'''
=== Sunject
*モチベーションの伝搬シミレーター
sim3_202xxxxx.py

====履歴
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 ３因子モデルに変更
*2020/11/14 メッセージの情報変容を導入

====利用論文

* 2020 IEEE Big Data Poster: An Analysis Approach of Messaging Mechanism on Social Networking Services20201110(One page)
* 2021 Inntelisys
* 2021 Black Sea

'''
import numpy as np
import matplotlib.pyplot as plt # import plot
import random                   # import random
import csv                      # read initial csv file

'''
Initial file
'''
def read_ini():     #Read initial file

    name = input('Enter the name of intial csv file name: ')
    row_val = []
    with open(name) as f:
        reader = csv.reader(f)
        for row in reader:
            row_val.append(row)
    return row_val

'''
Draw grapgh
'''
def draw_graph(l, ex, leg,g_num):   #draw the graph of the link range, l is a list, leg is the lagend of the graph
    print('I=',l)
    plt.subplot(1,3,int(g_num))
#   plt.plot(l, marker='o', Label='Exprience: No.{0}\n'.format(ex)+str(leg))
    plt.plot(l, marker='o', Label=str(leg))
    plt.legend()
    plt.xlabel('t in steps')
    plt.ylabel('I in Modification numbers')
    return

def draw_graph_all(l,r,tr,ex):    #All results
    fig, axes = plt.subplots(nrows = 1,ncols = 3)
    # fig.suptitle('Exprience: No.{0}\nSimulation:Q(t), R(t), Tr(t) (Motivation factors)'.format(ex_num))

    draw_graph(l, ex,legend+str('\nM_q'),1)
    draw_graph(r, ex,legend+str('\nM_r'),2)
```

```
    draw_graph(tr, ex,legend+str('\nM_tr'),3)
#    plt.show()
#    plt.savefig('result_fig{0}.png'.format(ex))
    return

'''
メッセージングモデルの定義

I(t)=[q(t), r(t), tr(t)]
メッセージはノードを超えるごとに加筆修正され伝搬すると仮定
    q(t)は、メッセージの内容で極性を持ち、情報量を示す
    r(t)は、コンテンツの信憑性（%）
    tr(t)は、メッセージを受けた相手の信頼性（コンテンツの信憑性とは別）（%）
    m(t)は、モチベーションで
        m(t) =1:メッセージをそのまま伝える
        m(t) =0:メッセージを伝えない
       0<m(t)<1: メッセージの変容度（%）
M_q(t)に　極性を持たせる
①(+)-(+): +1
②(-)-(-): +1
③(+)-(-): -1
④(-)-(+): -1
で①、②はこれまでの実験済
③、④が正規分布に頻度で発生するものとする

'''
'''
処理関数

'''
def messaging(information, message, threshold, m_ran_sw, m_ran_th_sw, m_q_sw,step):
#メッセージの発信

    qul = information[0]
    rho = information[1]
    tru = information[2]

    if m_ran_sw[0] == 'OFF':
        m_qul = message[0]
    else:
        m_qul = random.random()            #加筆修正をランダム化

    if m_ran_sw[1] == 'OFF':
        m_rho = message [1]
    else:
        m_rho = random.random()

    if m_ran_sw[2] == 'OFF':
        m_tru = message [2]
    else:
        m_tru = random.random()

    if m_q_sw == 'ON':
        m_qul = np.random.normal() * m_qul    #極性の正規分布配列
        m_list= [m_qul, m_rho, m_tru]
    else:
        m_list= [m_qul, m_rho, m_tru]

    print('In messaging m =', m_list)

```

```
    qul = qul * m_qul                           #メッセージの加筆修正
    rho = rho * m_rho
    tru = tru * m_tru

    information[0] = qul
    information[1] = rho
    information[2] = tru

    m_q_th = threshold[0]
    m_r_th = threshold[1]
    m_tr_th = threshold[2]

#    threshold[0] = m_q_th                       #しきい値のランダム化
    threshold[0] = random.random()* m_q_th

    if m_ran_th_sw[0] == 'OFF':
        threshold[0] = m_q_th
    else:                                       #加筆修正をランダム化
        if step != 0:
            m_q_th = random.random()*m_q_th
        else:
            if m_ran_th_sw[1] == 'OFF':
                threshold[1] = m_r_th
            else:
                if step !=0:
                    m_r_th = random.random()
                else:
                    if m_ran_th_sw[2] == 'OFF':
                        threshold[2] = m_tr_th
                    else:
                        if step != 0:
                            m_tr_th = random.random()
                        else:
                            threshold[0] = m_q_th
                            threshold[1] = m_r_th
                            threshold[2] = m_tr_th

    return information, m_list, threshold


def q_size_check(q_size, q_size_max):           #q(t)のサイズのチェック
    check_result = True

    if q_size > q_size_max:
        print('q_size =', q_size)
        check_result = False

    else:
        check_result = True

    return check_result


def message_check(mes,thr):                     #メッセージの修正加筆の大きさ
    modify = True

    m_quli = abs(mes[0])                        #

    m_rhoo = mes[1]
    m_trus = mes[2]
```

```
176
177     m_qu_th = thr[0]
178     m_rh_th = thr[1]
179     m_tru_th = thr[2]
180
181     if m_quli > m_qu_th:
182         print('m_quli = ', m_quli)
183         print('m_qu_th = ', m_qu_th)
184         modify = False
185     else:
186         if m_rhoo > m_rh_th:
187             print('m_rhoo = ', m_rhoo)
188             print('m_rh_th = ', m_rh_th)
189             modify = False
190         else:
191             if m_trus > m_tru_th:
192                 print('m_trus = ', m_trus)
193                 print('m_tru_th = ', m_tru_th)
194                 modify = False
195             else:
196                 modify = True
197     return modify
198
199 def decision_check(mes, maxx, maxi, step):                    #受信したメッセージのいくつかを見て発信するか否か？
200     dec = True
201
202  #    new_mes = sorted(mes, reverse = True)       #受信したメッセージの中で最大のq(t)を取り出す
203
204     print('mes =', mes)
205     print('maxx =', maxx)
206     print('maxi =', maxi)
207     print('step=', step)
208
209     m_quli = mes[0]
210     m_rhoo = mes[1]
211     m_trus = mes[2]
212
213     print('m_quli =', m_quli)
214     print('m_rhoo =', m_rhoo)
215     print('m_trus =', m_trus)
216
217     m_qu_max = maxx[0]
218     m_rh_max = maxx[1]
219     m_tru_max = maxx[2]
220
221     print('m_qu_max =', m_qu_max)
222     print('m_rh_max=', m_rh_max)
223     print('m_tru_max =', m_tru_max)
224
225
226     if abs(m_quli) > m_qu_max:          #
227         m_qu_max = m_quli
228         maxx[0] = m_q_max
229         maxi = step
230         dec = False
231     elif m_rhoo > m_rh_max:
232         m_rh_max = m_rhoo
233         maxx[1]= m_rh_max
```

```
        maxi = step
        dec = False
    elif m_trus > m_tru_max:
        m_tru_max = m_trus
        maxx[2]=m_tru_max
        maxi = step
        dec = False
    else:
        maxi = step
        dec = True
        print('Max check is ',dec)

    return dec, maxx, maxi

'''
############ main #############

'''
if __name__ == '__main__':
    row = read_ini()             #Read intial values from sim_init_xxxx
    print('Row =',row)
#
    ex_num = int(input('Enter the number of exprience [1-]: '))
    ex_num_index = ex_num-1

#
    kai= int(row[ex_num_index][23])     ## 履歴回数
    H =[]
#
    print('Number of events, kai= ',kai)
    rk = range (0,kai,1)
    m_th_q = float(row[ex_num_index][12])
    m_th_r = float(row[ex_num_index][13])
    m_th_tr = float(row[ex_num_index][14])
#
    m_th = [m_th_q,m_th_r,m_th_tr]      #Node(a)がもつしきい値群
    print('m_th =', m_th)


#
for k in rk:                 #イベント回数
    H = [m_th]                   #履歴を格納
    print ('H =',H)

    st = int(row[ex_num_index][1])
    q = float(row[ex_num_index][2])
    r = float(row[ex_num_index][3])
    tr = float(row[ex_num_index][4])

    m_q = float(row[ex_num_index][5])
    m_r = float(row[ex_num_index][6])
    m_tr = float(row[ex_num_index][7])

    q_max = float(row[ex_num_index][8]) #q_max is positive

    m_q_max = float(row[ex_num_index][9])
    m_r_max = float(row[ex_num_index][10])
    m_tr_max = float(row[ex_num_index][11])

    m_q_ran = row[ex_num_index][15]        #Random switch is str
```

```
    m_r_ran = row[ex_num_index][16]
    m_tr_ran = row[ex_num_index][17]

    m_ran = [m_q_ran, m_r_ran, m_tr_ran]     #Randam m swttch list
    print('M_ran sw are ',m_ran)

    m_q_th_ran = row[ex_num_index][18]        #Random switch is str
    m_r_th_ran = row[ex_num_index][19]
    m_tr_th_ran = row[ex_num_index][20]

    m_q_pn_sw = row[ex_num_index][21]       #P/N checker sw
    print('P/N swich is', m_q_pn_sw)

    m_ran_th = [m_q_th_ran, m_r_th_ran, m_tr_th_ran] #Random thresholds switch list

#    legend = row[ex_num_index][21]+str('\n==== Random variables ===\n')+str('Random for M_q is ')+m_q_ran+str('\nRandom for M_r is ')+m_r_ran+str('\nRandom for M_tr is ')+m_tr_ran
    legend = row[ex_num_index][22]

    print('st= ',st)
    rg = range (0,st,1)

    I = [q,r,tr]                #初期イベントEが出すメッセージ
    print('I =', I)

    m =[m_q, m_r, m_tr]          #Node(a)が出すメッセージ
    print('m =', m)

    print('q_max =', q_max)

    m_max =[m_q_max, m_r_max, m_tr_max] #Decisionの値群
    m_max_index = 0             #最大のs を  格納


    m_ran = [m_q_ran,m_r_ran,m_tr_ran]
    print('m_ran =', m_ran)


    L =[]                       #描画用のリスト格納&クリア
    M =[m]
    R =[]
    TR = []

# Main loop

    for s in rg:        #観測期間
        print ('s =',s)
        print('k =',k)
#
        m_th = H[s-1]             #履歴の読み込み
        print ('m_th =', m_th)
        print ('H =',H)
#
        message_prep = messaging(I, m, m_th, m_ran, m_ran_th, m_q_pn_sw,s)  #メッセージを発信準備
        I = message_prep[0] #メッセージを発信準備
        m = message_prep[1] #メッセージを発信準備
        m_th = message_prep[2]  #メッセージを発信準備
        print('Next I=',I)
```

```
        print('Before m=',m)

        decision = decision_check(m, m_max, m_max_index,s)
        m_max = decision[1]
        m_max_index = decision[2]
        print('Decision is ',decision[0])
        print('m_max=', m_max)
        print('m =',m)
        print('TR =',TR)

        if decision[0] == False: #メッセージを発信する意思はあるか？
            print('Massaing has terminated by decision in {0}
step(s).'.format(m_max_index-1))
            #if m_max_index == 0:
            #    break
            #else:
            print ('Termnated H =',H)
            draw_graph_all(L,R,TR,ex_num)
            break
        elif message_check(m,m_th) == False:  #メッセージの修正加筆がしきい値内か？
            print('Massaing has terminated by threshold over in {0}
step(s).'.format(m_max_index-1))
            #if m_max_index == 0:
            #    break
            #else:
            print ('Termnated H =',H)
            draw_graph_all(L,R,TR,ex_num)
            break
        elif q_size_check(abs(I[0]),q_max) == False: #メッセージが　最大数を超えていない
か？
            print('Massaing has terminated by size of messages in {0}
step(s).'.format(m_max_index-1))
            #if m_max_index == 0:
            #    break
            #else:
            print ('Termnated H =',H)
            draw_graph_all(L,R,TR,ex_num)
            break

        M.append(m)             #メッセージを追加

        print('After append m=', m)
        print('After append I=', I)
        print('========END of Kai=======')
        L.append(I[0])              #次のNodeへメッセージを渡す
        R.append(I[1])              #次のNodeへメッセージを渡す
        TR.append(I[2])              #次のNodeへメッセージを渡す
        H.append(m_th)               #履歴の記録
    else:
        draw_graph_all(L,R,TR,ex_num)
        print('M =',M)
else:
    plt.show()

```

● 3.2.7 パーソナライズ（レベル5）での嵩上げ（５０％）のソースコード

```
'''
=== Sunject
*モチベーションの伝搬シミレーター
sim3_202xxxxx.py

====履歴
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 ３因子モデルに変更
*2020/11/14 メッセージの情報変容を導入

====利用論文

* 2020 IEEE Big Data Poster: An Analysis Approach of Messaging Mechanism on Social Networking Services20201110(One page)
* 2021 Inntelisys
* 2021 Black Sea

'''
import numpy as np
import matplotlib.pyplot as plt # import plot
import random                   # import random
import csv                      # read initial csv file

'''
Initial file
'''
def read_ini():     #Read initial file
    global Qu, Re, Tru
    Qu = []                     #Multi drawing用
    Re = []
    Tru = []

    name = input('Enter the name of intial csv file name: ')
    row_val = []
    with open(name) as f:
        reader = csv.reader(f)
        for row in reader:
            row_val.append(row)
    return row_val

'''
Draw grapgh
'''
def draw_graph_qu(leg,kai_max):
    Q = []
    kai = 0
    while kai <=  kai_max:
        if type(Qu[kai]) is int:
            print ('整数のQ=',Q)
            Q.append(Qu[kai+1])
            plt.plot(Qu[kai], Label='Event #'+str(kai), marker='o')
            kai = kai+1
        else:
            Q.append(Qu[kai])
            print('途中 Qu=',Q)
            plt.plot(Qu[kai], Label='Event #'+str(kai), marker='o')
            kai = kai+1
    else:
```

```
        print('Q_multi = ',Q)
        print('kai =',kai)
        plt.title('Exprience: No.{0}\nSimulation:Behaviors of Q(t)\nEvents are {1} time.\n(Motivation factors\n'.format(ex_num, kai_max)+str(leg)+' Note: 50% raised)')

        plt.legend()
        plt.xlabel('t in steps')
        plt.ylabel('I in Modification numbers')
        plt.show()

    return



def draw_graph(ll, ex, leg,g_num):   #draw the graph of the link range, l is a list, leg is the lagend of the graph
#    Qu.append(ll)                  マルチグラフ用に格納



    plt.subplot(1,3,int(g_num))
#   plt.plot(l, marker='o', Label='Exprience: No.{0}\n'.format(ex)+str(leg))
    if g_num ==1:
        Qu.append(ll)
        print ('Qu =',Qu)
        plt.plot(ll, marker='o', Label=str(leg))
    elif g_num ==2:
        plt.plot(ll, marker='x', Label=str(leg))
    elif g_num ==3:
        plt.plot(ll, marker='*', Label=str(leg))
    else:
        return

def draw_graph_all(l,r,tr,ex,k,k_max):    #All results
    fig, axes = plt.subplots(nrows = 1,ncols = 3)
    fig.suptitle('Exprience: No.{0}\nSimulation:Q(t), R(t), Tr(t) (Motivation factors)'.format(ex_num))

    draw_graph(l, ex,legend+str('\nM_q'),1)
    #draw_graph(r, ex,legend+str('\nM_r'),2)
    #draw_graph(tr, ex,legend+str('\nM_tr'),3)
    plt.legend()
    plt.xlabel('t in steps')
    plt.ylabel('I in Modification numbers')
    return

'''
メッセージングモデルの定義

I(t)=[q(t), r(t), tr(t)]
メッセージはノードを超えるごとに加筆修正され伝搬すると仮定
    q(t)は、メッセージの内容で極性を持ち、情報量を示す
    r(t)は、情報コンテンツの信頼性（%）
    tr(t)は、メッセージを受けた相手の信用度（情報の信頼性とは別）（%）
    m(t)は、モチベーションで
        m(t) =1:メッセージをそのまま伝える
        m(t) =0:メッセージを伝えない
       0<m(t)<1: メッセージの変容度（%）
M_q(t)に  極性を持たせる
①(+)-(+): +1
```

```
②(-)-(-): +1
③(+)-(-): -1
④(-)-(+): -1
で①、②はこれまでの実験済
③、④が正規分布に頻度で発生するものとする

'''
'''
処理関数

'''
def messaging(information, message, threshold, m_ran_sw, m_ran_th_sw, m_q_sw,step):
#メッセージの発信

    qul = information[0]
    rho = information[1]
    tru = information[2]

    if m_ran_sw[0] == 'OFF':
        m_qul = message[0]
    else:
        m_qul = random.random()            #加筆修正をランダム化

    if m_ran_sw[1] == 'OFF':
        m_rho = message [1]
    else:
        m_rho = random.random()

    if m_ran_sw[2] == 'OFF':
        m_tru = message [2]
    else:
        m_tru = random.random()

    if m_q_sw == 'ON':
        m_qul = np.random.normal() * m_qul    #極性の正規分布配列
        m_list= [m_qul, m_rho, m_tru]
    else:
        m_list= [m_qul, m_rho, m_tru]

    print('In messaging m_list =', m_list)

    qul = qul * m_qul                              #メッセージの加筆修正
    rho = rho * m_rho
    tru = tru * m_tru



    information[0] = qul
    information[1] = rho
    information[2] = tru

    m_q_th = threshold[0]
    m_r_th = threshold[1]
    m_tr_th = threshold[2]

#    threshold[0] = m_q_th                        #しきい値のランダム化
#   threshold[0] = randam.random()

    if m_ran_th_sw[0] == 'OFF':
        threshold[0] = m_q_th
```

```
    else:                                           #加筆修正をランダム化
        if step != 0:                               #初期でランダムが発生しないように修正
            m_q_th = random.random()*m_q_th+0.5     #かさ上げ
        else:
            if m_ran_th_sw[1] == 'OFF':
                threshold[1] = m_r_th
            else:
                if step !=0:
                    m_r_th = random.random()
                else:
                    if m_ran_th_sw[2] == 'OFF':
                        threshold[2] = m_tr_th+0.5  #かさ上げ
                    else:
                        if step != 0:
                            m_tr_th = random.random()
                        else:
                            threshold[0] = m_q_th
                            threshold[1] = m_r_th
                            threshold[2] = m_tr_th

    return information, m_list, threshold

def motivation__check(mess,m_min):            #q(t)のサイズのチェック
    check_r = True

    if mess[1] < m_min:
        print('R =', mess[1])
        check_r = False
    elif mess[2] < m_min:
        print('Trust =', mess[2])
        check_r = False
    else:
        check_r = True

    return check_r


def q_size_check(q_size, q_size_max):            #q(t)のサイズのチェック
    check_result = True

    if q_size > q_size_max:
        print('q_size =', q_size)
        check_result = False

    else:
        check_result = True

    return check_result


def message_check(mes,thr):                      #メッセージの修正加筆の大きさ
    modify = True

    m_quli = abs(mes[0])                         #

    m_rhoo = mes[1]
    m_trus = mes[2]

    m_qu_th = thr[0]
    m_rh_th = thr[1]
```

```
    m_tru_th = thr[2]

    if m_quli > m_qu_th:
        print('m_quli = ', m_quli)
        print('m_qu_th = ', m_qu_th)
        modify = False
    else:
        if m_rhoo > m_rh_th:
            print('m_rhoo = ', m_rhoo)
            print('m_rh_th = ', m_rh_th)
            modify = False
        else:
            if m_trus > m_tru_th:
                print('m_trus = ', m_trus)
                print('m_tru_th = ', m_tru_th)
                modify = False
            else:
                modify = True
    return modify

def decision_check(mes, maxx, maxi, step):                    #受信したメッセージのいくつかを見て発信するか否か？
    dec = True

 #   new_mes = sorted(mes, reverse = True)       #受信したメッセージの中で最大のq(t)を取り出す

    print('mes =', mes)
    print('maxx =', maxx)
    print('maxi =', maxi)
    print('step=', step)

    m_quli = mes[0]
    m_rhoo = mes[1]
    m_trus = mes[2]

    print('m_quli =', m_quli)
    print('m_rhoo =', m_rhoo)
    print('m_trus =', m_trus)

    m_qu_max = maxx[0]
    m_rh_max = maxx[1]
    m_tru_max = maxx[2]

    print('m_qu_max =', m_qu_max)
    print('m_rh_max=', m_rh_max)
    print('m_tru_max =', m_tru_max)


    if abs(m_quli) > m_qu_max:          #
        m_qu_max = m_quli
        maxx[0] = m_q_max
        maxi = step
        dec = False
    elif m_rhoo > m_rh_max:
        m_rh_max = m_rhoo
        maxx[1]= m_rh_max
        maxi = step
        dec = False
    elif m_trus > m_tru_max:
```

```
        m_tru_max = m_trus
        maxx[2]=m_tru_max
        maxi = step
        dec = False
    else:
        maxi = step
        dec = True
        print('Max check is ',dec)

    return dec, maxx, maxi

'''
############ main #############

'''
if __name__ == '__main__':
    row = read_ini()             #Read intial values from sim_init_xxxx
    print('Row =',row)
#
    ex_num = int(input('Enter the number of exprience [1-]: '))
    ex_num_index = ex_num-1

#
    k_max= int(row[ex_num_index][23])     ## 履歴回数
    H =[]
#
    print('Number of events, k_max= ',k_max)
    rk = range (0,k_max,1)
    m_th_q = float(row[ex_num_index][12])
    m_th_r = float(row[ex_num_index][13])
    m_th_tr = float(row[ex_num_index][14])
#
    m_th = [m_th_q,m_th_r,m_th_tr]      #Node(a)がもつしきい値群
    print('m_th =', m_th)


#
for k in rk:                 #イベント回数
    H = [m_th]                   #履歴を格納
    print ('H =',H)
    Qu.append(k)
    print ('Qu =',Qu)


    st = int(row[ex_num_index][1])
    q = float(row[ex_num_index][2])
    r = float(row[ex_num_index][3])
    tr = float(row[ex_num_index][4])

    m_q = float(row[ex_num_index][5])
    m_r = float(row[ex_num_index][6])
    m_tr = float(row[ex_num_index][7])

    q_max = float(row[ex_num_index][8]) #q_max is positive

    m_q_max = float(row[ex_num_index][9])
    m_r_max = float(row[ex_num_index][10])
    m_tr_max = float(row[ex_num_index][11])

    m_q_ran = row[ex_num_index][15]        #Random switch is str
```

```
    m_r_ran = row[ex_num_index][16]
    m_tr_ran = row[ex_num_index][17]

    m_ran = [m_q_ran, m_r_ran, m_tr_ran]     #Randam m swttch list
    print('M_ran sw are ',m_ran)

    m_q_th_ran = row[ex_num_index][18]        #Random switch is str
    m_r_th_ran = row[ex_num_index][19]
    m_tr_th_ran = row[ex_num_index][20]

    m_q_pn_sw = row[ex_num_index][21]       #P/N checker sw
    print('P/N swich is', m_q_pn_sw)

    m_ran_th = [m_q_th_ran, m_r_th_ran, m_tr_th_ran] #Random thresholds switch list

#    legend = row[ex_num_index][21]+str('\n==== Random variables ===\n')+str('Random for M_q is ')+m_q_ran+str('\nRandom for M_r is ')+m_r_ran+str('\nRandom for M_tr is ')+m_tr_ran
    legend = row[ex_num_index][22]

    print('st= ',st)
    rg = range (0,st,1)

    I = [q,r,tr]                #初期イベントEが出すメッセージ
    print('I =', I)

    m =[m_q, m_r, m_tr]          #Node(a)が出すメッセージ
    print('m =', m)

    print('q_max =', q_max)

    m_max =[m_q_max, m_r_max, m_tr_max] #Decisionの値群
    m_max_index = 0             #最大のs を  格納


    m_ran = [m_q_ran,m_r_ran,m_tr_ran]
    print('m_ran =', m_ran)


    L =[]                       #描画用のリスト格納&クリア
    M =[m]
    R =[]
    TR = []

# Main loop

    for s in rg:        #観測期間
        print ('s =',s)
        print('k =',k)
#
        m_th = H[s-1]             #履歴の読み込み
        print ('m_th =', m_th)
        print ('H =',H)
#
        message_prep = messaging(I, m, m_th, m_ran, m_ran_th, m_q_pn_sw,s)  #メッセージを発信準備
        I = message_prep[0] #メッセージを発信準備
        m = message_prep[1] #メッセージを発信準備
        m_th = message_prep[2]  #メッセージを発信準備
        print('Next I=',I)
```

```
410         print('Before m=',m)
411 
412         decision = decision_check(m, m_max, m_max_index,s)
413         m_max = decision[1]
414         m_max_index = decision[2]
415         print('Decision is ',decision[0])
416         print('m_max=', m_max)
417         print('m =',m)
418         print('TR =',TR)
419 
420         if decision[0] == False: #メッセージを発信する意思はあるか？
421             print('Massaing has terminated by decision in {0}
    step(s).'.format(m_max_index-1))
422             print ('Termnated H =',H)
423             draw_graph_all(L,R,TR,ex_num,k,k_max)
424             break
425         elif message_check(m,m_th) == False:  #メッセージの修正加筆がしきい値内か？
426             print('Massaing has terminated by threshold over in {0}
    step(s).'.format(m_max_index-1))
427             print ('Termnated H =',H)
428             draw_graph_all(L,R,TR,ex_num,k,k_max)
429             break
430         elif q_size_check(abs(I[0]),q_max) == False: #メッセージが  最大数を超えていない
    か？
431             print('Massaing has terminated by size of messages in {0}
    step(s).'.format(m_max_index-1))
432             print ('Termnated H =',H)
433             draw_graph_all(L,R,TR,ex_num,k,k_max)
434             break
435         elif motivation__check(I, 0.01) == False: #モチベーションが低すぎないか？
436             print('Massaing has terminated by size of motivations in {0}
    step(s).'.format(m_max_index-1))
437             print ('Termnated H =',H)
438             draw_graph_all(L,R,TR,ex_num,k,k_max)
439             break
440 
441         M.append(m)             #メッセージを追加
442         print('After append m=', m)
443         print('After append I=', I)
444         print('========END of Kai========')
445         L.append(I[0])              #次のNodeへメッセージを渡す
446         R.append(I[1])              #次のNodeへメッセージを渡す
447         TR.append(I[2])              #次のNodeへメッセージを渡す
448 
449         H.append(m_th)              #履歴の記録
450     else:
451         draw_graph_all(L,R,TR,ex_num,k,k_max)
452         print('M =',M)
453 else:
454     plt.show()
455     draw_graph_qu(legend,k_max)
456 
```

付録２：第４章のシミュレーションのソースコード

- ４.５.１　シミュレーターの検証と集団戦略２集団のノードの動きと寝返り率のソースコード

```
'''
=== Subject
*モチベーションのシミレーター（Active nodes)の導入
sim_202xxxxx.py

====履歴
*2021/06/01 Active nodesを導入
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 ３因子モデルに変更
*2020/11/14 メッセージの情報変容を導入

====利用論文
'''

# モジュールのインポート
import random
import numpy as np
import matplotlib.pyplot as plt

# グローバル変数
N = 100          # Active nodesの個数
TIMELIMIT = 50  # シミュレーション打ち切り時刻
SEED = 65535     # 乱数の種
R = 0.5          # 近隣を規定する数値
# trust_level = 1.0     # Party_1のActive nodesの初期信頼性の強さ
AREA = 25        # 描画範囲
PAUSE_TIME = 0.001      # 描画間隔
# クラス定義
# Nodeクラス
class Node:
    """Active nodesを表現するクラスの定義"""
    def __init__(self, sta):  # コンストラクタ
        self.status = sta
        self.x = (random.random() - 0.5) * AREA  # x座標の初期値
        self.y = (random.random() - 0.5) * AREA  # y座標の初期値
    def calcnext(self):  # 次時刻の状態の計算
        if self.status == 0:
            self.sta0()  # Party_0の計算
        elif self.status == 1:
            self.sta1()  # Party_1の計算
        else:  # 合致するPartyがない
            print("ERROR PArtyがありません\n")
    def sta0(self):  # Party_0の計算メソッド
        # Party_1のActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status == 1:
                c0x = self.x
                c0y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c0x - ax) * (c0x - ax) + (c0y - ay) * (c0y - ay)) < R:
                # 隣接してParty_1のActive nodesがいる
                    self.status = 1  # Party_1に変身
        # 位置の更新
        self.x += random.random() - 0.5
        self.y += random.random() - 0.5
    def sta1(self):  # Party_1の計算メソッド
        self.x += (random.random() - 0.5) * trust_level
```

```
        self.y += (random.random() - 0.5) * trust_level
    def putstate(self):  # 状態の出力
        print(self.status, self.x, self.y)
# Nodeクラスの定義の終わり

# 関数の定義
# calcn()関数
def calcn(a):
    """次時刻の状態を計算"""
    for i in range(len(a)):
        a[i].calcnext()
        a[i].putstate()

        # グラフデータに現在位置を追加
        if a[i].status == 0:
            xlist0.append(a[i].x)
            ylist0.append(a[i].y)
        elif a[i].status == 1:
            xlist1.append(a[i].x)
            ylist1.append(a[i].y)
# calcn()関数の終わり
# 描画関係の関数
# draw_nodes Active nodeの動きを観察
def draw_nodes(xli0, yli0, xli1, yli1, reali, cread, ti, cou0, cou1, sum_ch):
    plt.clf()  # グラフ領域のクリア
    plt.axis([AREA*(-1), AREA, AREA*(-1), AREA])   # 描画領域の設定（暫定）
    plt.title('### Exprience: Party expantion###\nRealiablity level = {0}%,  Creadit level = {1}\n Time = {2}\nParty 0 has {3} members\nParty 1 has {4} members \
\nSum of change = {5}'.format(reali * 100,cread, ti, cou0, cou1, sum_ch))
    plt.plot(xli0, yli0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
    plt.plot(xli1, yli1, "*", color='red', label = 'Party 1')  # Party_1をプロット
    plt.legend()
    plt.xlabel("Diffusion distance(x)")
    plt.ylabel("Diffusion distance(y)")
    plt.legend()
    plt.grid()
    plt.pause(PAUSE_TIME)
# draw_nodes関数の終わり
# draw_trust 別のpartyにいくつ反転させたか
def draw_trust(changeli, reali, cread):
    plt.plot(changeli)
    plt.title('### Exprience: Party expantion###\nRealiablity level = {0}%,  Creadit level = {1}'.format(reali * 100,cread))
    plt.xlabel("Time (steps)")
    plt.ylabel("Number of status change (times)")
    plt.grid()
# draw_trust関数の終わり
# input_trust Realiablity levelとCreadit levelの入力
def input_trust():
    rev0 = float(input("Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1）："))
    while (rev0< 0) or (rev0 > 1):
        rev0 = float(input("入力が範囲を越えています。再度Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1）："))
    rev1 = float(input("Party_1のActive nodesの同じpartyのメンバーに対する信用性（Credibility）を入力してください。（0-5）："))
    while (rev1 < 0) or (rev1 > 5):
        rev1 = float(input("入力が範囲を越えています。再度Party_1のActive nodesの同じpartyのメンバーに対する信用性（Credibility）を入力してください。（0-5）："))
    return rev0,rev1
```

```
# input_trustの終わり

################
# メインループ
################
if __name__ == "__main__":
    random.seed(SEED)  # 乱数の初期化
    # Party_0のActive nodesの生成
    a = [Node(0) for i in range(N)]
# Party_1のActive nodesを設定、座業を原点からずらす
    a[0].status = 1
    a[0].x = -2
    a[0].y = -2
# Party_1のActive nodesの[Active nodesの信頼性]の設定
    realiable_level,creadit_level  = input_trust()
    trust_level =realiable_level * creadit_level
# グラフデータの初期化
# Party_0のデータ
    xlist0 = []
    ylist0 = []
# Party_1のデータ
    xlist1 = []
    ylist1 = []
# 反転回数
    changelist = []
# Active nodesシミュレーション
    for t in range(TIMELIMIT):
        calcn(a)  # 次時刻の状態を計算
        count0 = len(xlist0)
        count1 = len(xlist1)
        sum_change = count1 -1
        changelist.append(sum_change)
        draw_nodes(xlist0, ylist0, xlist1, ylist1,realiable_level,
creadit_level,t,count0, count1, sum_change)
            # 描画データのクリア
        xlist0.clear()
        ylist0.clear()
        xlist1.clear()
        ylist1.clear()
    plt.show()

    # 反転回数の描画
    draw_trust(changelist, realiable_level, creadit_level)
    plt.show()
#  メインループの終わり

```

● 4.5.3 影響度 R を変える ソースコード

```
'''
=== Subject
*モチベーションのシミレーター（Active nodes)の導入
sim_202xxxxx.py

====履歴
*2021/06/01 Active nodesを導入
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 ３因子モデルに変更
*2020/11/14 メッセージの情報変容を導入

====利用論文
*


====利用について
コメントアウトすれば機能がする内容は #XXXXXX
で示してあります

'''

# モジュールのインポート
import random
import numpy as np
import matplotlib.pyplot as plt

# グローバル変数
N = 100          # Active nodesの総個数
M = int(N/2)     # Party 1に設定
TIMELIMIT = 100  # シミュレーション打ち切り時刻
SEED = 65535     # 乱数の種
# SEED = 7
# R = 0.2          # 近隣を規定する数値
# trust_level = 1.0     # Party_1のActive nodesの初期信頼性の強さ
AREA = 25        # Agentの動き観察用描画範囲
PAUSE_TIME = 0.001      # 描画間隔
EX = 1000        # 実験回数
STEP = 0.001     # Rの変化のステップ数
CMAP = "hsv"     # 複合グラフの色空間
# クラス定義
# Nodeクラス
class Node:
    """Active nodesを表現するクラスの定義"""
    def __init__(self, sta):  # コンストラクタ
        self.status = sta
        self.x = (random.random() - 0.5) * AREA  # x座標の初期値
        self.y = (random.random() - 0.5) * AREA  # y座標の初期値
    def calcnext(self):  # 次時刻の状態の計算
        if self.status == 0:
            self.sta0()  # Party_0の計算
        elif self.status == 1:
            self.sta1()  # Party_1の計算
        else:  # 合致するPartyがない
            print("ERROR PArtyがありません\n")
    def sta0(self):  # Party_0の計算メソッド
        # Party_1のActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status == 1:
```

```
                c0x = self.x
                c0y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c0x - ax) * (c0x - ax) + (c0y - ay) * (c0y - ay)) < R:
                # 隣接してParty_1のActive nodesがいる
                    self.status = 1  # Party_1に変身
        # 位置の更新
        self.x += random.random() - 0.5
        self.y += random.random() - 0.5
    def sta1(self):  # Party_1の計算メソッド
        self.x += (random.random() - 0.5) * trust_level
        self.y += (random.random() - 0.5) * trust_level
    def putstate(self):  # 状態の出力
        print(self.status, self.x, self.y)
# Nodeクラスの定義の終わり

# 関数の定義
# calcn()関数
def calcn(a):
    """次時刻の状態を計算"""
    for i in range(len(a)):
        a[i].calcnext()
        # a[i].putstate()

        # グラフデータに現在位置を追加
        if a[i].status == 0:
            xlist0.append(a[i].x)
            ylist0.append(a[i].y)
        elif a[i].status == 1:
            xlist1.append(a[i].x)
            ylist1.append(a[i].y)
# calcn()関数の終わり
# 描画関係の関数
# draw_nodes Active nodeの動きを観察
def draw_nodes(xli0, yli0, xli1, yli1, reali, cread, ti, cou0, cou1, sum_ch,exn,
ran):
    plt.clf()  # グラフ領域のクリア
    plt.axis([AREA*(-1), AREA, AREA*(-1), AREA])   # 描画領域の設定（暫定）
    plt.title('# Exprience: Party expantion #\nRealiablity level = {0}%,  Creadit
level = {1}\n Time = {2}\nParty 0 has {3} members     Party 1 has {4} members \
\nSum of change = {5}%\nNumber of Trial = {6}   Range R = {7}'.format(reali * 100,
cread, ti, cou0, cou1, sum_ch, exn+1, ran))
    plt.plot(xli0, yli0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
    plt.plot(xli1, yli1, "*", color='red', label = 'Party 1')  # Party_1をプロット
    plt.xlabel("Diffusion distance(x)")

    plt.ylabel("Diffusion distance(y)")
    plt.legend()
    plt.grid()
    plt.pause(PAUSE_TIME)
# draw_nodes関数の終わり
def draw_R_sum(r, s,n,m, ex):
    plt.title('# Exprience: Party expantion #\nR - chane %\nN ={0}, M ={1}, Exprience
= {2} (times)'.format(n,m,ex))
    plt.plot(r,s)  # Rと変更%をプロット
    ax = plt.gca()
    ax.set_xscale('log')
    ax.set_yscale('log')
    plt.xlabel("Level (R)")
```

```
    plt.ylabel("% of change nodes")
    plt.grid(color ='black')
# draw_R_sum関数の終わり
# draw_trust 別のpartyにいくつ反転させたか
def draw_trust(etime,changeli, reali, cread, cmap):
    plt.plot(changeli, color= cmap(etime/10))
    plt.title('Realiablity level = {0}%,  Creadit level = {1}\nNumber of Exprience = {2} (times)'.format(reali * 100, cread, etime))
    plt.grid(color ='gray')
    plt.xlabel("Time (steps)")
    plt.ylabel("Number of status change (times)")

# draw_trust関数の終わり
# input_trust Realiablity levelとCreadit levelの入力
def input_trust():
    rev0 = float(input("Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
    while (rev0< 0) or (rev0 > 1):
        rev0 = float(input("入力が範囲を越えています。再度Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
    rev1 = float(input("Party_1のActive nodesの同じpartyのメンバーに対する信用性（Credibility）を入力してください。（0-5)）: "))
    while (rev1 < 0) or (rev1 > 5):
        rev1 = float(input("入力が範囲を越えています。再度Party_1のActive nodesの同じpartyのメンバーに対する信用性（Credibility）を入力してください。（0-5)）: "))
    rev2 = float(input("Active nodesの間の影響度(R)を入力してください。（0-5)）: "))
    while (rev2 < 0) or (rev2 > 5):
        rev2 = float(input("入力が範囲を越えています。再度Active nodesの間の影響度(R)を入力してください。（0-5)）: "))
    return rev0,rev1,rev2
# input_trustの終わり

################
# メインループ
################
if __name__ == "__main__":
    # Party_1のActive nodesの[Active nodesの信頼性]の設定
    realiable_level,creadit_level, R  = input_trust()
    trust_level =realiable_level * creadit_level
    random.seed(SEED)  # 乱数の初期化
    cm = plt.get_cmap(CMAP)
    Rlist = []
    sum_change_per_list = []
    # random.random()
    for et in range(EX):
        # Party_0のActive nodesの生成
        a = [Node(0) for i in range(N)]
    # Party_1のActive nodesを設定、座業を原点からずらす
    # a[0].status = 1
        a[0].x = -2
        a[0].y = -2
    # Party 1をM個
        for m in range(M - 1):
            a[m].status = 1

    # グラフデータの初期化
    # Party_0のデータ
        xlist0 = []
        ylist0 = []
    # Party_1のデータ
```

```
        xlist1 = []
        ylist1 = []
    # 反転回数
        changelist = []
    # Active nodesシミュレーション
        for t in range(TIMELIMIT):
            calcn(a)  # 次時刻の状態を計算
            count0 = len(xlist0)
            count1 = len(xlist1)
            sum_change = count1 - M
            changelist.append(sum_change)
            sum_change_per = sum_change/N *100
            #draw_nodes(xlist0, ylist0, xlist1, ylist1,realiable_level,
creadit_level,t,count0, count1, sum_change_per,et, R)
                # 描画データのクリア
            xlist0.clear()
            ylist0.clear()
            xlist1.clear()
            ylist1.clear()
        # plt.show()
        #    print ('Number of experiments is ', et)
        #    draw_trust(et, changelist, realiable_level, creadit_level,cm) # 反復回数
        # changelist.clear()
        Rlist.append(R)
        sum_change_per_list.append(sum_change_per)
        draw_R_sum(Rlist, sum_change_per_list,N, M, EX)
        R += STEP
    plt.show()
#  メインループの終わり

```

- ４.５.３　影響度 R を変える　２集団のノード分布を個別で表示するソースコード

```
'''
=== Subject
*モチベーションのシミレーター（Active nodes)の導入
sim_202xxxxx.py

====履歴
*2021/06/01 Active nodesを導入
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 ３因子モデルに変更
*2020/11/14 メッセージの情報変容を導入

====利用論文
*


====利用について
コメントアウトすれば機能がする内容は #XXXXXX
で示してあります

'''

# モジュールのインポート
import random
import numpy as np
import matplotlib.pyplot as plt

# グローバル変数
N = 100          # Active nodesの総個数
M = int(N/2)     # Party 1に設定
TIMELIMIT = 100  # シミュレーション打ち切り時刻
# SEED = 65535     # 乱数の種
SEED = 7
# R = 0.2          # 近隣を規定する数値
# trust_level = 1.0     # Party_1のActive nodesの初期信頼性の強さ
AREA = 25        # Agentの動き観察用描画範囲
PAUSE_TIME = 0.001      # 描画間隔
EX = 100          # 実験回数
STEP = 0.001      # Rの変化のステップ数
CMAP = "hsv"     # 複合グラフの色空間
# クラス定義
# Nodeクラス
class Node:
    """Active nodesを表現するクラスの定義"""
    def __init__(self, sta):  # コンストラクタ
        self.status = sta
        self.x = (random.random() - 0.5) * AREA  # x座標の初期値
        self.y = (random.random() - 0.5) * AREA  # y座標の初期値
    def calcnext(self):  # 次時刻の状態の計算
        if self.status == 0:
            self.sta0()  # Party_0の計算
        elif self.status == 1:
            self.sta1()  # Party_1の計算
        else:  # 合致するPartyがない
            print("ERROR PArtyがありません\n")
    def sta0(self):  # Party_0の計算メソッド
        # Party_1のActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status == 1:
```

```
                c0x = self.x
                c0y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c0x - ax) * (c0x - ax) + (c0y - ay) * (c0y - ay)) < R:
                # 隣接してParty_1のActive nodesがいる
                    self.status = 1  # Party_1に変身
        # 位置の更新
        self.x += random.random() - 0.5
        self.y += random.random() - 0.5
    def sta1(self):  # Party_1の計算メソッド
        self.x += (random.random() - 0.5) * trust_level
        self.y += (random.random() - 0.5) * trust_level
    def putstate(self):  # 状態の出力
        print(self.status, self.x, self.y)
# Nodeクラスの定義の終わり

# 関数の定義
# calcn()関数
def calcn(a):
    """次時刻の状態を計算"""
    for i in range(len(a)):
        a[i].calcnext()
        # a[i].putstate()

        # グラフデータに現在位置を追加
        if a[i].status == 0:
            xlist0.append(a[i].x)
            ylist0.append(a[i].y)
        elif a[i].status == 1:
            xlist1.append(a[i].x)
            ylist1.append(a[i].y)
# calcn()関数の終わり
# 描画関係の関数
# draw_nodes Active nodeの動きを観察
def draw_nodes(xli0, yli0, xli1, yli1, reali, cread, ti, cou0, cou1, sum_ch,exn,
ran):
    plt.clf()  # グラフ領域のクリア
    plt.axis([AREA*(-1), AREA, AREA*(-1), AREA])   # 描画領域の設定（暫定）
    plt.title('# Exprience: Party expantion #\nRealiablity level = {0}%,  Creadit
level = {1}\n Time = {2}\nParty 0 has {3} members     Party 1 has {4} members \
\nSum of change = {5}%\nNumber of Trial = {6}   Range R = {7}'.format(reali * 100,
cread, ti, cou0, cou1, sum_ch, exn+1, ran))
    plt.plot(xli0, yli0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
    plt.plot(xli1, yli1, "*", color='red', label = 'Party 1')  # Party_1をプロット
    plt.xlabel("Diffusion distance(x)")

    plt.ylabel("Diffusion distance(y)")
    plt.legend()
    plt.grid()
    plt.pause(PAUSE_TIME)
# draw_nodes関数の終わり
def draw_R_sum(r, s):
    plt.title('# Exprience: Party expantion #\nR - chane %')
    plt.plot(r,s)  # Rと変更%をプロット
    plt.xlabel("Level (R)")
    plt.ylabel("% of change nodes")
    plt.grid(color ='gray')
# draw_R_sum関数の終わり
# draw_trust 別のpartyにいくつ反転させたか
```

```
118 def draw_trust(etime,changeli, reali, cread, cmap):
119     plt.plot(changeli, color= cmap(etime/10))
120     plt.title('Realiablity level = {0}%,  Creadit level = {1}\nNumber of Exprience =
    {2} (times)'.format(reali * 100, cread, etime))
121     plt.grid(color ='gray')
122     plt.xlabel("Time (steps)")
123     plt.ylabel("Number of status change (times)")
124 
125 # draw_trust関数の終わり
126 # input_trust Realiablity levelとCreadit levelの入力
127 def input_trust():
128     rev0 = float(input("Party_1のActive nodesのメッセージの真偽性（realiability）を入力
    してください。（0-1）: "))
129     while (rev0< 0) or (rev0 > 1):
130         rev0 = float(input("入力が範囲を越えています。再度Party_1のActive nodesのメッセー
    ジの真偽性（realiability）を入力してください。（0-1）: "))
131     rev1 = float(input("Party_1のActive nodesの同じpartyのメンバーに対する信用性
    （Credibility）を入力してください。（0-5）: "))
132     while (rev1 < 0) or (rev1 > 5):
133         rev1 = float(input("入力が範囲を越えています。再度Party_1のActive nodesの同じ
    partyのメンバーに対する信用性（Credibility）を入力してください。（0-5）: "))
134     rev2 = float(input("Active nodesの間の影響度(R)を入力してください。（0-5）: "))
135     while (rev2 < 0) or (rev2 > 5):
136         rev2 = float(input("入力が範囲を越えています。再度Active nodesの間の影響度(R)を入
    力してください。（0-5）: "))
137     return rev0,rev1,rev2
138 # input_trustの終わり
139 
140 ################
141 # メインループ
142 ################
143 if __name__ == "__main__":
144     # Party_1のActive nodesの[Active nodesの信頼性]の設定
145     realiable_level,creadit_level, R  = input_trust()
146     trust_level =realiable_level * creadit_level
147     random.seed(SEED)  # 乱数の初期化
148     cm = plt.get_cmap(CMAP)
149     Rlist = []
150     sum_change_per_list = []
151     # random.random()
152     for et in range(EX):
153         # Party_0のActive nodesの生成
154         a = [Node(0) for i in range(N)]
155     # Party_1のActive nodesを設定、座業を原点からずらす
156     # a[0].status = 1
157         a[0].x = -2
158         a[0].y = -2
159     # Party 1をM個
160         for m in range(M - 1):
161             a[m].status = 1
162 
163     # グラフデータの初期化
164     # Party_0のデータ
165         xlist0 = []
166         ylist0 = []
167     # Party_1のデータ
168         xlist1 = []
169         ylist1 = []
170     # 反転回数
171         changelist = []
```

```
    # Active nodesシミュレーション
        for t in range(TIMELIMIT):
            calcn(a)  # 次時刻の状態を計算
            count0 = len(xlist0)
            count1 = len(xlist1)
            sum_change = count1 - M
            changelist.append(sum_change)
            sum_change_per = sum_change/N *100
            #draw_nodes(xlist0, ylist0, xlist1, ylist1,realiable_level,
creadit_level,t,count0, count1, sum_change_per,et, R)
                # 描画データのクリア
            xlist0.clear()
            ylist0.clear()
            xlist1.clear()
            ylist1.clear()
        # plt.show()
        #    print ('Number of experiments is ', et)
        #    draw_trust(et, changelist, realiable_level, creadit_level,cm) # 反復回数
        # changelist.clear()
        Rlist.append(R)
        sum_change_per_list.append(sum_change_per)
        draw_R_sum(Rlist, sum_change_per_list)
        R += STEP
    plt.show()
#  メインループの終わり

```

● 4.5.3 影響度 R を変える 2集団のノード分布を示すソースコード

```
'''
=== Subject
*モチベーションのシミレーター（Active nodes)の導入
sim_202xxxxx.py

====履歴
*2021/06/01 Active nodesを導入
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 ３因子モデルに変更
*2020/11/14 メッセージの情報変容を導入

====利用論文
*


====利用について
コメントアウトすれば機能がする内容は #XXXXXX
で示してあります

'''

# モジュールのインポート
import random
import numpy as np
import matplotlib.pyplot as plt

# グローバル変数
N = 100          # Active nodesの個数
M = 50           # Party 1に設定
TIMELIMIT = 100  # シミュレーション打ち切り時刻
# SEED = 65535     # 乱数の種
SEED = 7
R = 0.2          # 近隣を規定する数値
# trust_level = 1.0     # Party_1のActive nodesの初期信頼性の強さ
AREA = 25        # 描画範囲
PAUSE_TIME = 0.001      # 描画間隔
EX = 1           # 実験回数
CMAP = "hsv" #
# クラス定義
# Nodeクラス
class Node:
    """Active nodesを表現するクラスの定義"""
    def __init__(self, sta):  # コンストラクタ
        self.status = sta
        self.x = (random.random() - 0.5) * AREA  # x座標の初期値
        self.y = (random.random() - 0.5) * AREA  # y座標の初期値
    def calcnext(self):  # 次時刻の状態の計算
        if self.status == 0:
            self.sta0()  # Party_0の計算
        elif self.status == 1:
            self.sta1()  # Party_1の計算
        else:  # 合致するPartyがない
            print("ERROR PArtyがありません\n")
    def sta0(self):  # Party_0の計算メソッド
        # Party_1のActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status == 1:
                c0x = self.x
```

```
                c0y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c0x - ax) * (c0x - ax) + (c0y - ay) * (c0y - ay)) < R:
                # 隣接してParty_1のActive nodesがいる
                    self.status = 1  # Party_1に変身
        # 位置の更新
        self.x += random.random() - 0.5
        self.y += random.random() - 0.5
    def sta1(self):  # Party_1の計算メソッド
        self.x += (random.random() - 0.5) * trust_level
        self.y += (random.random() - 0.5) * trust_level
    def putstate(self):  # 状態の出力
        print(self.status, self.x, self.y)
# Nodeクラスの定義の終わり

# 関数の定義
# calcn()関数
def calcn(a):
    """次時刻の状態を計算"""
    for i in range(len(a)):
        a[i].calcnext()
        # a[i].putstate()

        # グラフデータに現在位置を追加
        if a[i].status == 0:
            xlist0.append(a[i].x)
            ylist0.append(a[i].y)
        elif a[i].status == 1:
            xlist1.append(a[i].x)
            ylist1.append(a[i].y)
# calcn()関数の終わり
# 描画関係の関数
# draw_nodes Active nodeの動きを観察
def draw_nodes(xli0, yli0, xli1, yli1, reali, cread, ti, cou0, cou1, sum_ch,exn,
ran):
    plt.clf()  # グラフ領域のクリア
    plt.axis([AREA*(-1), AREA, AREA*(-1), AREA])   # 描画領域の設定（暫定）
    plt.title('# Exprience: Party expantion #\nRealiablity level = {0}%,  Creadit
level = {1}\n Time = {2}\nParty 0 has {3} members\nParty 1 has {4} members \
\nSum of change = {5}\nNumber of Trial = {6}\nRange R = {7}'.format(reali * 100,
cread, ti, cou0, cou1, sum_ch, exn+1, ran))
    plt.plot(xli0, yli0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
    plt.plot(xli1, yli1, "*", color='red', label = 'Party 1')  # Party_1をプロット
    plt.legend()
    plt.xlabel("Diffusion distance(x)")

    plt.ylabel("Diffusion distance(y)")
    plt.legend()
    plt.grid()
    plt.pause(PAUSE_TIME)
# draw_nodes関数の終わり
# draw_trust 別のpartyにいくつ反転させたか
def draw_trust(etime,changeli, reali, cread, cmap):
    plt.plot(changeli, color= cmap(etime/10))
    plt.title('Realiablity level = {0}%,  Creadit level = {1}\nNumber of Exprience =
{2} (times)'.format(reali * 100, cread, etime))
    plt.grid(color ='gray')
    plt.xlabel("Time (steps)")
    plt.ylabel("Number of status change (times)")
```

```
117
118 # draw_trust関数の終わり
119 # input_trust Realiablity levelとCreadit levelの入力
120 def input_trust():
121     rev0 = float(input("Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
122     while (rev0< 0) or (rev0 > 1):
123         rev0 = float(input("入力が範囲を越えています。再度Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
124     rev1 = float(input("Party_1のActive nodesの同じpartyのメンバーに対する信用性（Credibility）を入力してください。（0-5)）: "))
125     while (rev1 < 0) or (rev1 > 5):
126         rev1 = float(input("入力が範囲を越えています。再度Party_1のActive nodesの同じpartyのメンバーに対する信用性（Credibility）を入力してください。（0-5)）: "))
127     return rev0,rev1
128 # input_trustの終わり
129
130 ###############
131 # メインループ
132 ###############
133 if __name__ == "__main__":
134     # Party_1のActive nodesの[Active nodesの信頼性]の設定
135     realiable_level,creadit_level  = input_trust()
136     trust_level =realiable_level * creadit_level
137     random.seed(SEED)  # 乱数の初期化
138     cm = plt.get_cmap(CMAP)
139     # random.random()
140     for et in range(EX):
141         # Party_0のActive nodesの生成
142         a = [Node(0) for i in range(N)]
143     # Party_1のActive nodesを設定、座業を原点からずらす
144     # a[0].status = 1
145         a[0].x = -2
146         a[0].y = -2
147     # Party 1をM個
148         for m in range(M - 1):
149             a[m].status = 1
150
151     # グラフデータの初期化
152     # Party_0のデータ
153         xlist0 = []
154         ylist0 = []
155     # Party_1のデータ
156         xlist1 = []
157         ylist1 = []
158     # 反転回数
159         changelist = []
160     # Active nodesシミュレーション
161         for t in range(TIMELIMIT):
162             calcn(a)  # 次時刻の状態を計算
163             count0 = len(xlist0)
164             count1 = len(xlist1)
165             sum_change = count1 - M
166             changelist.append(sum_change)
167             draw_nodes(xlist0, ylist0, xlist1, ylist1,realiable_level,creadit_level,t,count0, count1, sum_change,et, R)
168                 # 描画データのクリア
169             xlist0.clear()
170             ylist0.clear()
171             xlist1.clear()
```

```
            ylist1.clear()
        # plt.show()
        #    print ('Number of experiments is ', et)
        #    draw_trust(et, changelist, realiable_level, creadit_level,cm) # 反復回数
        # changelist.clear()
    plt.show()
#  メインループの終わり

```

● 4.5.3 影響度 R を変える 連続して１次近似を行うソースコード

```
# -*- coding: utf-8 -*-
'''
=== Subject
*モチベーションのシミレーター（Active nodes)の導入
sim_202xxxxx.py
1
====履歴
*2021/08/09 Semi-logでの近似(First-order function approximation by least squares method)
*2021/06/01 Active nodesを導入
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 ３因子モデルに変更
*2020/11/14 メッセージの情報変容を導入

====利用論文
*2021/08/01 CCS  Es2021 abstract


====利用について
コメントアウトすれば機能がする内容は #XXXXXX
で示してあります

'''

# モジュールのインポート
from tqdm import tqdm  # Pregoress bar
import random
import numpy as np
import matplotlib.pyplot as plt

# グローバル変数
N = 100          # Active nodesの総個数
M = int(N/2)      # Party 1に設定　ここで半数に設定
TIMELIMIT = 100  # シミュレーション打ち切り時刻
SEED = 65535     # 乱数の種
# SEED = 7
# R = 0.2           # 近隣を規定する数値
# trust_level = 1.0     # Party_1のActive nodesの初期信頼性の強さ
AREA = 25         # Agentの動き観察用描画範囲
PAUSE_TIME = 0.001       # 描画間隔
EX = 1000         # 実験回数
STEP = 0.001      # Rの変化のステップ数
CMAP = "hsv"      # 複合グラフの色空間
STEP2 = 1       # 決定係数と回帰式の窓領域の変化を見る際の刻み
START_WIN = 1    # 窓領域の初期値
# START_WIN = 10    # 窓領域の初期値
START_END = 60    # 窓領域の最終値
1
# クラス定義
# Nodeクラス
class Node:
    """Active nodesを表現するクラスの定義"""
    def __init__(self, sta):  # コンストラクタ
        self.status = sta
        self.x = (random.random() - 0.5) * AREA  # x座標の初期値
        self.y = (random.random() - 0.5) * AREA  # y座標の初期値
    def calcnext(self):  # 次時刻の状態の計算
        if self.status == 0:
```

```
            self.sta0()  # Party_0の計算
        elif self.status == 1:
            self.sta1()  # Party_1の計算
        else:  # 合致するPartyがない
            print("ERROR PArtyがありません\n")
    def sta0(self):  # Party_0の計算メソッド
        # Party_1のActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status == 1:
                c0x = self.x
                c0y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c0x - ax) * (c0x - ax) + (c0y - ay) * (c0y - ay)) < R:
                # 隣接してParty_1のActive nodesがいる
                    self.status = 1  # Party_1に変身
        # 位置の更新
        self.x += random.random() - 0.5
        self.y += random.random() - 0.5
    def sta1(self):  # Party_1の計算メソッド
        self.x += (random.random() - 0.5) * trust_level
        self.y += (random.random() - 0.5) * trust_level
    def putstate(self):  # 状態の出力
        print(self.status, self.x, self.y)
# Nodeクラスの定義の終わり

# 関数の定義
# calcn()関数
def calcn(a):
    """次時刻の状態を計算"""
    for i in range(len(a)):
        a[i].calcnext()
        # a[i].putstate()

        # グラフデータに現在位置を追加
        if a[i].status == 0:
            xlist0.append(a[i].x)
            ylist0.append(a[i].y)
        elif a[i].status == 1:
            xlist1.append(a[i].x)
            ylist1.append(a[i].y)
# calcn()関数の終わり
# 描画関係の関数
# draw_nodes Active nodeの動きを観察
def draw_nodes(xli0, yli0, xli1, yli1, reali, cread, ti, cou0, cou1, sum_ch,exn,
ran):
    plt.clf()  # グラフ領域のクリア
    plt.axis([AREA*(-1), AREA, AREA*(-1), AREA])   # 描画領域の設定（暫定）
    plt.title('Realiablity level = {0}%,  Creadit level = {1}\n Time = {2}\nParty 0
has {3} members     Party 1 has {4} members \
\nSum of change = {5}%\nNumber of Trial = {6}   Range R = {7}'.format(reali * 100,
cread, ti, cou0, cou1, sum_ch, exn+1, ran))
    plt.plot(xli0, yli0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
    plt.plot(xli1, yli1, "*", color='red', label = 'Party 1')  # Party_1をプロット
    plt.xlabel("Diffusion distance(x)")

    plt.ylabel("Diffusion distance(y)")
    plt.legend()
    plt.grid()
    plt.pause(PAUSE_TIME)
```

```
# draw_nodes関数の終わり
# draw_R_sum関数　センシティブなRの描画
def draw_R_sum(r, s,n,m, ex):
    plt.title('R - chane %\nN ={0}, M ={1}, Exprience = {2} (times)'.format(n,m,ex))
    plt.plot(r,s)  # Rと変更%をプロット
    ax = plt.gca()
    ax.set_xscale('log')    # x軸対数表示
#    ax.set_yscale('log')   # y軸対数表示
    plt.xlabel("Level (R)")
    plt.ylabel("% of change nodes (S)")
    plt.grid(color ='black')
# draw_R_sum関数の終わり
# draw_trust 別のpartyにいくつ反転させたか
def draw_trust(etime,changeli, reali, cread, cmap):
    plt.plot(changeli, color= cmap(etime/10))
    plt.title('Realiablity level = {0}%,  Creadit level = {1}\nNumber of Exprience = {2} (times)'.format(reali * 100, cread, etime))
    plt.grid(color ='gray')
    plt.xlabel("Time (steps)")
    plt.ylabel("Number of status change (times)")
# draw_trust関数の終わり
# draw_appro(ximation) 全データに対してSemi-log10での1次関数近似(logR-Changing party)
def draw_appro(r,s):
    x = np.array(r)
    y = np.array(s)
    log_x = [np.log10(i) for i in x]    # 底を10
    linear = np.polyfit(log_x, y, 1)
    y_linear = [linear[0] * x_linear + linear[1] for x_linear in log_x]
    plt.plot(x, y_linear, label = '100%')
# draw_approの終わり
# draw_appro(ximation) part 一部のデータに対してSemi-log10での1次関数近似(logR-Changing party)
def draw_appro_part(r,s,step):
    leng_x = int(len(r) * step/100)
    r0 = r[:leng_x]
    s0 = s[:leng_x]
    x = np.array(r0)
    y = np.array(s0)
    log_x = [np.log10(i) for i in x]    # 底を10#
    linear = np.polyfit(log_x, y, 1)
    y_linear = [linear[0] * x_linear + linear[1] for x_linear in log_x]
    # 相関係数と決定係数を計算する
    coe = np.corrcoef(y, y_linear)[0,1] # 相関係数R
    coe2= coe ** 2 # 決定係数R^2
    print('coe(相関係数) =',coe,'coe2（決定係数） =',coe2)
    plt.plot(x, y_linear, label  = '{0} %\nS = {1} * log10 (R)\n + {2}\nCoe^2 = {3}'.format(step, linear[0], linear[1], coe2))
    return coe2 # 決定係数を返す
# draw_appro_partの終わり
# draw_appro(ximation) best 一部のデータに対してSemi-log10での最適な1次関数近似(logR-Changing party)
def draw_appro_best(r,s,wind_best,legend1, legend2, legend3):
    leng_x = int(len(r) * wind_best/100)
    r0 = r[:leng_x]
    s0 = s[:leng_x]
    x = np.array(r0)
    y = np.array(s0)
    log_x = [np.log10(i) for i in x]    # 底を10#
    linear = np.polyfit(log_x, y, 1)
    y_linear = [linear[0] * x_linear + linear[1] for x_linear in log_x]
```

```
    plt.title('Realiablity level = {0}%,  Creadit level = {1}, R = {2}'.format(legend1 * 100, legend2, legend3))
    plt.plot(x, y_linear, label  = 'At {0} %\nS = {1} * log10 (R)\n + {2}'.format(wind_best,linear[0], linear[1]))
    plt.legend()
# draw_appro_bestの終わり
# draw_R2 窓領域と決定係数のグラフ、戻り値：窓の最大値
def draw_R2(r):
    max2 = max(r)
    i = r.index(max(r))
    plt.plot(r, label = 'Max value ={0} at {1} (%)'.format(max2, i+1)) #最大値とインデックス＋１
    plt.grid(color ='gray')
    plt.xlabel("Window size (%)")
    plt.ylabel("Coefficent of determination")
    return i+1
# draw_R2の終1わり
# パラメーター入力関数
# input_trust Realiablity levelとCreadit level, 近似範囲の入力
def input_trust():
    rev0 = float(input("Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
    while (rev0< 0) or (rev0 > 1):
        rev0 = float(input("入力が範囲を越えています。再度Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
    rev1 = float(input("Party_1のActive nodesの同じpartyのメンバーに対する信憑性（Credibility）を入力してください。（0-5)）: "))
    while (rev1 < 0) or (rev1 > 5):
        rev1 = float(input("入力が範囲を越えています。再度Party_1のActive nodesの同じpartyのメンバーに対する信用性（Credibility）を入力してください。（0-5)）: "))
    rev2 = float(input("Active nodesの間の影響度(R)を入力してください。（0-5)）: "))
    while (rev2 < 0) or (rev2 > 5):
        rev2 = float(input("入力が範囲を越えています。再度Active nodesの間の影響度(R)を入力してください。（0-5)）: "))
    rev3 = float(input("影響度(R)の近似範囲を入力してください。（1-100 %)）: "))
    while (rev3 < 0) or (rev3 > 100):
        rev3 = float(input("入力が範囲を越えています。再度、近似範囲を入力してください。（1-100 %)）: "))
    return rev0,rev1,rev2,rev3
# input_trustの終わり

###############
# メインループ
###############
if __name__ == "__main__":
    # Party_1のActive nodesの[Active nodesの信頼性]の設定
    realiable_level, creadit_level, R, appro_area  = input_trust()
    trust_level =realiable_level * creadit_level
    r_ini = R   # 入力されたRを格納
    random.seed(SEED)  # 乱数の初期化
    cm = plt.get_cmap(CMAP)
    Rlist = []
    sum_change_per_list = []
    R2list = []          # 決定係数用
    # random.random()
#    for et in range(EX):
    for et in tqdm(range(EX)):          #Pregress  表示
        # Party_0のActive nodesの生成
        a = [Node(0) for i in range(N)]
    # Party_1のActive nodesを設定、座業を原点からずらす
```

```
    # a[0].status = 1
        a[0].x = -2
        a[0].y = -2
    # Party 1をM個
        for m in range(M - 1):
            a[m].status = 1

    # グラフデータの初期化
    # Party_0のデータ
        xlist0 = []
        ylist0 = []
    # Party_1のデータ
        xlist1 = []
        ylist1 = []
    # 反転回数
        changelist = []
    # Active nodesシミュレーション
        for t in range(TIMELIMIT):
            calcn(a)  # 次時刻の状態を計算
            count0 = len(xlist0)
            count1 = len(xlist1)
            sum_change = count1 - M
            changelist.append(sum_change)
            sum_change_per = sum_change/N *100
            #draw_nodes(xlist0, ylist0, xlist1, ylist1,realiable_level,
creadit_level,t,count0, count1, sum_change_per,et, R)
                # 描画データのクリア
            xlist0.clear()
            ylist0.clear()
            xlist1.clear()
            ylist1.clear()
        # plt.show()
        #    print ('Number of experiments is ', et)
        #    draw_trust(et, changelist, realiable_level, creadit_level,cm) # 反復回数
        # changelist.clear()
        Rlist.append(R)
        sum_change_per_list.append(sum_change_per)
        draw_R_sum(Rlist, sum_change_per_list,N, M, EX)
        R += STEP
    draw_appro(Rlist,sum_change_per_list)
    for k in range(START_WIN, START_END, STEP2):
        # draw_appro_part(Rlist,sum_change_per_list,appro_area)   #部分近似の窓領域を変
化
        decfact = draw_appro_part(Rlist,sum_change_per_list,k)   #部分近似の窓領域を変
化
        R2list.append(decfact)
    # plt.legend()
    plt.show()
    i_best = draw_R2(R2list)
    plt.legend()
    plt.show()
    draw_R_sum(Rlist, sum_change_per_list,N, M, EX) # Draw again + 部分近似
    draw_appro_best(Rlist,sum_change_per_list,i_best,realiable_level, creadit_level,
r_ini) # 最適近似1次直線近似
    plt.show()
    print('\n========== The End of program ===========')

#  メインループの終わり

```

● 4.6.1 同数の集団に対する第3集団で個別に表示するソースコード

```
# -*- coding: utf-8 -*-
'''
=== Subject
*モチベーションのシミレーター（Active nodes)の導入
sim_202xxxxx.py

====履歴
*2021/09/05 P3ノードの導入と改変
*2021/08/09 Semi-logでの近似(First-order function approximation by least squares method)
*2021/06/01 Active nodesを導入
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 ３因子モデルに変更
*2020/11/14 メッセージの情報変容を導入

====利用論文
*2021/08/01 CCS  Es2021 abstract


====利用について
コメントアウトすれば機能がする内容は #XXXXXX
で示してあります

'''

# モジュールのインポート
import random
import numpy as np
import matplotlib.pyplot as plt

# グローバル変数
# N = 100          # Active nodesの総個数
# M = int(N/2)     # Party 1に設定　ここで半数に設定
M = 50            # Party 1に設定
P = 50            # Party 0に設定
Q = 10             # Party 2に設定

N = M + P + Q     # Active nodesの総個


TIMELIMIT = 50  # シミュレーション打ち切り時刻
SEED = 65535      # 乱数の種
# SEED = 7
# R = 0.2           # 近隣を規定する数値
# trust_level = 1.0     # Party_1のActive nodesの初期信頼性の強さ
AREA = 25         # Agentの動き観察用描画範囲
PAUSE_TIME = 0.001       # 描画間隔
EX = 10           # 実験回数
STEP = 0.001      # Rの変化のステップ数
CMAP = "hsv"      # 複合グラフの色空間
#
# クラス定義
# Nodeクラス
class Node:
    """Active nodesを表現するクラスの定義"""
    def __init__(self, sta):  # コンストラクタ
        self.status = sta
        self.x = (random.random() - 0.5) * AREA  # x座標の初期値
```

```
        self.y = (random.random() - 0.5) * AREA  # y座標の初期値
    def calcnext(self):  # 次時刻の状態の計算
        if self.status == 0:
            self.sta0()  # Party_0の計算
        elif self.status == 1:
            self.sta1()  # Party_1の計算
        elif self.status == 2:
            self.sta2()  # Party_2の計算(第3勢力)
        else:  # 合致するPartyがない
            print("ERROR Partyがありません\n")
    def sta0(self):  # Party_0の計算メソッド
        # Party_1のActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 0:
                c0x = self.x
                c0y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c0x - ax) * (c0x - ax) + (c0y - ay) * (c0y - ay)) < R:
                    if a[i].status == 1:    # 隣接してParty_1のActive nodesがいる
                        self.status = 1  # Party_1に変身
                    elif a[i].status == 2:  # 隣接してParty_2のActive nodesがいる
                        self.status = 2  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level

    def sta1(self):  # Party_1の計算メソッド
        # 他のPartyのActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 1:
                c1x = self.x
                c1y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c1x - ax) * (c1x - ax) + (c1y - ay) * (c1y - ay)) < R:
                    if a[i].status == 0:    # 隣接してParty_0のActive nodesがいる
                        self.status = 0  # Party_0に変身
                    elif a[i].status == 2:  # 隣接してParty_2のActive nodesがいる
                        self.status = 2  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level

    def sta2(self):  # Party_2の計算メソッド
        # 他のPartyのActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 2:
                c2x = self.x
                c2y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c2x - ax) * (c2x - ax) + (c2y - ay) * (c2y - ay)) < R:
                    if a[i].status == 0:    # 隣接してParty_0のActive nodesがいる
                        self.status = 0  # Party_0に変身
                    elif a[i].status == 1:  # 隣接してParty_1のActive nodesがいる
                        self.status = 1  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level
```

```
120
121     def putstate(self):  # 状態の出力
122         print(self.status, self.x, self.y)
123 # Nodeクラスの定義の終わり
124 # 関数の定義
125 # calcn()関数
126 def calcn(a):
127     """次時刻の状態を計算"""
128     for i in range(len(a)):
129         a[i].calcnext()
130         # a[i].putstate()
131 
132         # グラフデータに現在位置を追加
133         if a[i].status == 0:
134             xlist0.append(a[i].x)
135             ylist0.append(a[i].y)
136         elif a[i].status == 1:
137             xlist1.append(a[i].x)
138             ylist1.append(a[i].y)
139         elif a[i].status == 2:
140             xlist2.append(a[i].x)
141             ylist2.append(a[i].y)
142 # calcn()関数の終わり
143 #
144 # 描画関係の関数
145 # draw_nodes: Active nodeの動きを観察
146 def draw_nodes(xli0, yli0, xli1, yli1,xli2, yli2, reali, cread, ti, cou0, cou1, cou2,
    sum_ch,exn, ran):
147     plt.clf()  # グラフ領域のクリア
148     plt.axis([AREA*(-1), AREA, AREA*(-1), AREA])   # 描画領域の設定（暫定）
149     plt.title('Realiablity level = {0}%,  Creadit level = {1}\n Time = {2}\nParty 0
    has {3} members, Party 1 has {4}, members Party 2 has {5} members\
150 \nSum of change = {6}%\nNumber of Trial = {7}   Range R = {8}'.format(reali * 100,
    cread, ti, cou0, cou1, cou2,sum_ch, exn+1, ran))
151     plt.plot(xli0, yli0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
152     plt.plot(xli1, yli1, "*", color='red', label = 'Party 1')  # Party_1をプロット
153     plt.plot(xli2, yli2, "x", color='green', label = 'Party 2')  # Party_2をプロット
154     plt.xlabel("Diffusion distance(x)")
155     plt.ylabel("Diffusion distance(y)")
156     plt.legend()
157     plt.grid()
158     plt.pause(PAUSE_TIME)
159 # draw_nodes関数の終わり
160 # draw_count: Active nodeの動きを観察
161 def draw_count(reali, cread, cou0, cou1, cou2,e,ran, num):
162     plt.title('Realiablity level = {0}%, Creadit level = {1}\nRange R = {2}, No. of
    Ex = {3}, Total Nodes ={4}'.format(reali * 100, cread,ran, e+1, num))
163     plt.grid()
164     plt.plot(cou0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
165     plt.plot(cou1, "*", color='red', label = 'Party 1')  # Party_1をプロット
166     plt.plot(cou2, "x", color='green', label = 'Party 2')  # Party_2をプロット
167     # plt.legend()
168     plt.xlabel("Time (steps)")
169     plt.ylabel("Numbers of nodes in each party")
170     print('Drawing nou: {0}'.format(e))
171 # draw_count関数の終わり
172 
173 # パラメーター入力関数
174 # input_trust Realiablity levelとCreadit level, 近似範囲の入力
175 def input_trust():
```

```
    rev0 = float(input("Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
    while (rev0< 0) or (rev0 > 1):
        rev0 = float(input("入力が範囲を越えています。再度Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
    rev1 = float(input("Party_1のActive nodesの同じpartyのメンバーに対する信憑性（Credibility）を入力してください。（0-5)）: "))
    while (rev1 < 0) or (rev1 > 5):
        rev1 = float(input("入力が範囲を越えています。再度Party_1のActive nodesの同じpartyのメンバーに対する信用性（Credibility）を入力してください。（0-5)）: "))
    rev2 = float(input("Active nodesの間の影響度(R)を入力してください。（0-5)）: "))
    while (rev2 < 0) or (rev2 > 5):
        rev2 = float(input("入力が範囲を越えています。再度Active nodesの間の影響度(R)を入力してください。（0-5)）: "))
    return rev0,rev1,rev2
# input_trustの終わり

###############
# メインループ
###############
if __name__ == "__main__":
    # Party_1のActive nodesの[Active nodesの信頼性]の設定
    realiable_level, creadit_level, R = input_trust()
    trust_level =realiable_level * creadit_level
    r_ini = R   # 入力されたRを格納
    random.seed(SEED)  # 乱数の初期化
    cm = plt.get_cmap(CMAP)

    for et in range(EX):
        # Party_0のActive nodesの生成
        a = [Node(0) for i in range(N)]
    # Party_1のActive nodesを設定、座業を原点からずらす
        a[0].status = 0
        a[0].x = -2
        a[0].y = -2
    # Party 1をP個
        for p in range(P):
            a[p].status = 0
    # Party 1をM個
        for m in range(M):
            a[m].status = 1
    # Party 2をQ個
        for n in range(Q):
            a[n].status = 2

    # グラフデータの初期化
    # Party_0のデータ
        xlist0 = []
        ylist0 = []
    # Party_1のデータ
        xlist1 = []
        ylist1 = []
    # Party_2のデータ
        xlist2 = []
        ylist2 = []
    # ノード数
        sta0_sum = []   #Party 0のノード数の結果
        sta1_sum = []   #Party 1のノード数の結果
        sta2_sum = []   #Party 2のノード数の結果

```

```
    # 反転回数
        changelist = []
    # Active nodesシミュレーション
        for t in range(TIMELIMIT):
            calcn(a)  # 次時刻の状態を計算
            count0 = len(xlist0)
            count1 = len(xlist1)
            count2 = len(xlist2)
            sum_change = count1 - M
            changelist.append(sum_change)
            sum_change_per = sum_change/N *100
            # draw_nodes(xlist0, ylist0, xlist1, ylist1,xlist2,
ylist2,realiable_level, creadit_level,t,count0, count1, count2,sum_change_per,et, R)
            # 描画データのクリア
            xlist0.clear()
            ylist0.clear()
            xlist1.clear()
            ylist1.clear()
            xlist2.clear()
            ylist2.clear()
            # 各ノードの総数
            sta0_sum.append(count0)   #Party 0のノード数の結果
            sta1_sum.append(count1)    #Party 1のノード数の結果
            sta2_sum.append(count2)    #Party 2のノード数の結果
        draw_count(realiable_level, creadit_level,sta0_sum, sta1_sum, sta2_sum, et,
R, N)
    #plt.legend(['Party 0','Party 1','Party 2'])
        plt.show() ####### 個別表示を標示

    print('\n========= The End of program ===========')

#  メインループの終わり

```

- ４.６.２　同数の集団に対する第３のノードのＲの影響で数回の試行をまとめ表示するソースコード

```
# -*- coding: utf-8 -*-
'''
=== Subject
*モチベーションのシミレーター（Active nodes)の導入
sim_202xxxxx.py

====履歴
*2021/09/05 P3ノードの導入と改変
*2021/08/09 Semi-logでの近似(First-order function approximation by least squares method)
*2021/06/01 Active nodesを導入
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 ３因子モデルに変更
*2020/11/14 メッセージの情報変容を導入

====利用論文
*2021/08/01 CCS  Es2021 abstract


====利用について
コメントアウトすれば機能がする内容は #XXXXXX
で示してあります

'''

# モジュールのインポート
import random
import numpy as np
import matplotlib.pyplot as plt

# グローバル変数
# N = 100          # Active nodesの総個数
# M = int(N/2)     # Party 1に設定　ここで半数に設定
M = 10           # Party 1に設定
P = 90           # Party 0に設定
Q = 0            # Party 2に設定

N = M + P + Q    # Active nodesの総個


TIMELIMIT = 50  # シミュレーション打ち切り時刻
SEED = 65535     # 乱数の種
# SEED = 7
# R = 0.2          # 近隣を規定する数値
# trust_level = 1.0     # Party_1のActive nodesの初期信頼性の強さ
AREA = 25        # Agentの動き観察用描画範囲
PAUSE_TIME = 0.001       # 描画間隔
EX = 10          # 実験回数
STEP = 0.001     # Rの変化のステップ数
CMAP = "hsv"     # 複合グラフの色空間
#
# クラス定義
# Nodeクラス
class Node:
    """Active nodesを表現するクラスの定義"""
    def __init__(self, sta):  # コンストラクタ
        self.status = sta
        self.x = (random.random() - 0.5) * AREA  # x座標の初期値
```

```
        self.y = (random.random() - 0.5) * AREA  # y座標の初期値
    def calcnext(self):  # 次時刻の状態の計算
        if self.status == 0:
            self.sta0()  # Party_0の計算
        elif self.status == 1:
            self.sta1()  # Party_1の計算
        elif self.status == 2:
            self.sta2()  # Party_2の計算(第3勢力)
        else:  # 合致するPartyがない
            print("ERROR Partyがありません\n")
    def sta0(self):  # Party_0の計算メソッド
        # Party_1のActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 0:
                c0x = self.x
                c0y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c0x - ax) * (c0x - ax) + (c0y - ay) * (c0y - ay)) < R:
                    if a[i].status == 1:    # 隣接してParty_1のActive nodesがいる
                        self.status = 1  # Party_1に変身
                    elif a[i].status == 2:  # 隣接してParty_2のActive nodesがいる
                        self.status = 2  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level

    def sta1(self):  # Party_1の計算メソッド
        # 他のPartyのActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 1:
                c1x = self.x
                c1y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c1x - ax) * (c1x - ax) + (c1y - ay) * (c1y - ay)) < R:
                    if a[i].status == 0:    # 隣接してParty_0のActive nodesがいる
                        self.status = 0  # Party_0に変身
                    elif a[i].status == 2:  # 隣接してParty_2のActive nodesがいる
                        self.status = 2  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level

    def sta2(self):  # Party_2の計算メソッド
        # 他のPartyのActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 2:
                c2x = self.x
                c2y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c2x - ax) * (c2x - ax) + (c2y - ay) * (c2y - ay)) < R:
                    if a[i].status == 0:    # 隣接してParty_0のActive nodesがいる
                        self.status = 0  # Party_0に変身
                    elif a[i].status == 1:  # 隣接してParty_1のActive nodesがいる
                        self.status = 1  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level
```

```
120
121     def putstate(self):  # 状態の出力
122         print(self.status, self.x, self.y)
123 # Nodeクラスの定義の終わり
124 # 関数の定義
125 # calcn()関数
126 def calcn(a):
127     """次時刻の状態を計算"""
128     for i in range(len(a)):
129         a[i].calcnext()
130         # a[i].putstate()
131 
132         # グラフデータに現在位置を追加
133         if a[i].status == 0:
134             xlist0.append(a[i].x)
135             ylist0.append(a[i].y)
136         elif a[i].status == 1:
137             xlist1.append(a[i].x)
138             ylist1.append(a[i].y)
139         elif a[i].status == 2:
140             xlist2.append(a[i].x)
141             ylist2.append(a[i].y)
142 # calcn()関数の終わり
143 #
144 # 描画関係の関数
145 # draw_nodes: Active nodeの動きを観察
146 def draw_nodes(xli0, yli0, xli1, yli1,xli2, yli2, reali, cread, ti, cou0, cou1, cou2,
    sum_ch,exn, ran):
147     plt.clf()  # グラフ領域のクリア
148     plt.axis([AREA*(-1), AREA, AREA*(-1), AREA])   # 描画領域の設定（暫定）
149     plt.title('Realiablity level = {0}%,  Creadit level = {1}\n Time = {2}\nParty 0
    has {3} members, Party 1 has {4}, members Party 2 has {5} members\
150 \nSum of change = {6}%\nNumber of Trial = {7}   Range R = {8}'.format(reali * 100,
    cread, ti, cou0, cou1, cou2,sum_ch, exn+1, ran))
151     plt.plot(xli0, yli0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
152     plt.plot(xli1, yli1, "*", color='red', label = 'Party 1')  # Party_1をプロット
153     plt.plot(xli2, yli2, "x", color='green', label = 'Party 2')  # Party_2をプロット
154     plt.xlabel("Diffusion distance(x)")
155     plt.ylabel("Diffusion distance(y)")
156     plt.legend()
157     plt.grid()
158     plt.pause(PAUSE_TIME)
159 # draw_nodes関数の終わり
160 # draw_count: Active nodeの動きを観察
161 def draw_count(reali, cread, cou0, cou1, cou2,e,ran, num):
162     plt.title('Realiablity level = {0}%, Creadit level = {1}\nRange R = {2}, No. of
    Ex = {3}, Total Nodes ={4}'.format(reali * 100, cread,ran, e+1, num))
163     plt.grid()
164     plt.plot(cou0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
165     plt.plot(cou1, "*", color='red', label = 'Party 1')  # Party_1をプロット
166     plt.plot(cou2, "x", color='green', label = 'Party 2')  # Party_2をプロット
167     # plt.legend()
168     plt.xlabel("Time (steps)")
169     plt.ylabel("Numbers of nodes in each party")
170     print('Drawing nou: {0}'.format(e))
171 # draw_count関数の終わり
172 
173 # パラメーター入力関数
174 # input_trust Realiablity levelとCreadit level, 近似範囲の入力
175 def input_trust():
```

```
    rev0 = float(input("Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
    while (rev0< 0) or (rev0 > 1):
        rev0 = float(input("入力が範囲を越えています。再度Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
    rev1 = float(input("Party_1のActive nodesの同じpartyのメンバーに対する信憑性（Credibility）を入力してください。（0-5)）: "))
    while (rev1 < 0) or (rev1 > 5):
        rev1 = float(input("入力が範囲を越えています。再度Party_1のActive nodesの同じpartyのメンバーに対する信用性（Credibility）を入力してください。（0-5)）: "))
    rev2 = float(input("Active nodesの間の影響度(R)を入力してください。（0-5)）: "))
    while (rev2 < 0) or (rev2 > 5):
        rev2 = float(input("入力が範囲を越えています。再度Active nodesの間の影響度(R)を入力してください。（0-5)）: "))
    return rev0,rev1,rev2
# input_trustの終わり

###############
# メインループ
###############
if __name__ == "__main__":
    # Party_1のActive nodesの[Active nodesの信頼性]の設定
    realiable_level, creadit_level, R = input_trust()
    trust_level =realiable_level * creadit_level
    r_ini = R   # 入力されたRを格納
    random.seed(SEED)  # 乱数の初期化
    cm = plt.get_cmap(CMAP)

    for et in range(EX):
        # Party_0のActive nodesの生成
        a = [Node(0) for i in range(N)]
    # Party_1のActive nodesを設定、座業を原点からずらす
        a[0].status = 0
        a[0].x = -2
        a[0].y = -2
    # Party 1をP個
        for p in range(P):
            a[p].status = 0
    # Party 1をM個
        for m in range(M):
            a[m].status = 1
    # Party 2をQ個
        for n in range(Q):
            a[n].status = 2

    # グラフデータの初期化
    # Party_0のデータ
        xlist0 = []
        ylist0 = []
    # Party_1のデータ
        xlist1 = []
        ylist1 = []
    # Party_2のデータ
        xlist2 = []
        ylist2 = []
    # ノード数
        sta0_sum = []   #Party 0のノード数の結果
        sta1_sum = []   #Party 1のノード数の結果
        sta2_sum = []   #Party 2のノード数の結果

```

```
    # 反転回数
        changelist = []
    # Active nodesシミュレーション
        for t in range(TIMELIMIT):
            calcn(a)  # 次時刻の状態を計算
            count0 = len(xlist0)
            count1 = len(xlist1)
            count2 = len(xlist2)
            sum_change = count1 - M
            changelist.append(sum_change)
            sum_change_per = sum_change/N *100
            # draw_nodes(xlist0, ylist0, xlist1, ylist1,xlist2,
ylist2,realiable_level, creadit_level,t,count0, count1, count2,sum_change_per,et, R)
            # 描画データのクリア
            xlist0.clear()
            ylist0.clear()
            xlist1.clear()
            ylist1.clear()
            xlist2.clear()
            ylist2.clear()
            # 各ノードの総数
            sta0_sum.append(count0)   #Party 0のノード数の結果
            sta1_sum.append(count1)    #Party 1のノード数の結果
            sta2_sum.append(count2)    #Party 2のノード数の結果
        draw_count(realiable_level, creadit_level,sta0_sum, sta1_sum, sta2_sum, et,
R, N)
    plt.legend(['Party 0','Party 1','Party 2'])
    plt.show() ####### まとめ図を標示
    print('\n========== The End of program ===========')

#  メインループの終わり

```

- ４.６.２　同数の集団に対する第３のノードのＲの影響で第３集団が１ノードの場合の拡散の様子を示すソースコード

```
# -*- coding: utf-8 -*-
'''
=== Subject
*モチベーションのシミレーター（Active nodes)の導入
sim_202xxxxx.py

====履歴
*2021/09/05 P3ノードの導入と改変
*2021/08/09 Semi-logでの近似(First-order function approximation by least squares method)
*2021/06/01 Active nodesを導入
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 ３因子モデルに変更
*2020/11/14 メッセージの情報変容を導入

====利用論文
*2021/08/01 CCS  Es2021 abstract


====利用について
コメントアウトすれば機能がする内容は #XXXXXX
で示してあります

'''

# モジュールのインポート
import random
import numpy as np
import matplotlib.pyplot as plt

# グローバル変数
# N = 100          # Active nodesの総個数
# M = int(N/2)     # Party 1に設定　ここで半数に設定
M = 50            # Party 1に設定
P = 50            # Party 0に設定
Q = 1             # Party 2に設定

N = M + P + Q     # Active nodesの総個


TIMELIMIT = 50  # シミュレーション打ち切り時刻
SEED = 65535      # 乱数の種
# SEED = 7
# R = 0.2           # 近隣を規定する数値
# trust_level = 1.0     # Party_1のActive nodesの初期信頼性の強さ
AREA = 25         # Agentの動き観察用描画範囲
PAUSE_TIME = 0.001       # 描画間隔
EX = 10           # 実験回数
STEP = 0.001      # Rの変化のステップ数
CMAP = "hsv"      # 複合グラフの色空間
#
# クラス定義
# Nodeクラス
class Node:
    """Active nodesを表現するクラスの定義"""
    def __init__(self, sta):  # コンストラクタ
        self.status = sta
        self.x = (random.random() - 0.5) * AREA  # x座標の初期値
```

```
        self.y = (random.random() - 0.5) * AREA  # y座標の初期値
    def calcnext(self):  # 次時刻の状態の計算
        if self.status == 0:
            self.sta0()  # Party_0の計算
        elif self.status == 1:
            self.sta1()  # Party_1の計算
        elif self.status == 2:
            self.sta2()  # Party_2の計算(第3勢力)
        else:  # 合致するPartyがない
            print("ERROR Partyがありません\n")
    def sta0(self):  # Party_0の計算メソッド
        # Party_1のActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 0:
                c0x = self.x
                c0y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c0x - ax) * (c0x - ax) + (c0y - ay) * (c0y - ay)) < R:
                    if a[i].status == 1:    # 隣接してParty_1のActive nodesがいる
                        self.status = 1  # Party_1に変身
                    elif a[i].status == 2:  # 隣接してParty_2のActive nodesがいる
                        self.status = 2  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level

    def sta1(self):  # Party_1の計算メソッド
        # 他のPartyのActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 1:
                c1x = self.x
                c1y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c1x - ax) * (c1x - ax) + (c1y - ay) * (c1y - ay)) < R:
                    if a[i].status == 0:    # 隣接してParty_0のActive nodesがいる
                        self.status = 0  # Party_0に変身
                    elif a[i].status == 2:  # 隣接してParty_2のActive nodesがいる
                        self.status = 2  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level

    def sta2(self):  # Party_2の計算メソッド
        # 他のPartyのActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 2:
                c2x = self.x
                c2y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c2x - ax) * (c2x - ax) + (c2y - ay) * (c2y - ay)) < R:
                    if a[i].status == 0:    # 隣接してParty_0のActive nodesがいる
                        self.status = 0  # Party_0に変身
                    elif a[i].status == 1:  # 隣接してParty_1のActive nodesがいる
                        self.status = 1  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level
```

```
120
121     def putstate(self):  # 状態の出力
122         print(self.status, self.x, self.y)
123 # Nodeクラスの定義の終わり
124 # 関数の定義
125 # calcn()関数
126 def calcn(a):
127     """次時刻の状態を計算"""
128     for i in range(len(a)):
129         a[i].calcnext()
130         # a[i].putstate()
131 
132         # グラフデータに現在位置を追加
133         if a[i].status == 0:
134             xlist0.append(a[i].x)
135             ylist0.append(a[i].y)
136         elif a[i].status == 1:
137             xlist1.append(a[i].x)
138             ylist1.append(a[i].y)
139         elif a[i].status == 2:
140             xlist2.append(a[i].x)
141             ylist2.append(a[i].y)
142 # calcn()関数の終わり
143 #
144 # 描画関係の関数
145 # draw_nodes: Active nodeの動きを観察
146 def draw_nodes(xli0, yli0, xli1, yli1,xli2, yli2, reali, cread, ti, cou0, cou1, cou2,
    sum_ch,exn, ran):
147     plt.clf()  # グラフ領域のクリア
148     plt.axis([AREA*(-1), AREA, AREA*(-1), AREA])   # 描画領域の設定（暫定）
149     plt.title('Realiablity level = {0}%,  Creadit level = {1}\n Time = {2}\nParty 0
    has {3} members, Party 1 has {4}, members Party 2 has {5} members\
150 \nSum of change = {6}%\nNumber of Trial = {7}   Range R = {8}'.format(reali * 100,
    cread, ti, cou0, cou1, cou2,sum_ch, exn+1, ran))
151     plt.plot(xli0, yli0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
152     plt.plot(xli1, yli1, "*", color='red', label = 'Party 1')  # Party_1をプロット
153     plt.plot(xli2, yli2, "x", color='green', label = 'Party 2')  # Party_2をプロット
154     plt.xlabel("Diffusion distance(x)")
155     plt.ylabel("Diffusion distance(y)")
156     plt.legend()
157     plt.grid()
158     plt.pause(PAUSE_TIME)
159 # draw_nodes関数の終わり
160 # draw_count: Active nodeの動きを観察
161 def draw_count(reali, cread, cou0, cou1, cou2,e,ran, num):
162     plt.title('Realiablity level = {0}%, Creadit level = {1}\nRange R = {2}, No. of
    Ex = {3}, Total Nodes ={4}'.format(reali * 100, cread,ran, e+1, num))
163     plt.grid()
164     plt.plot(cou0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
165     plt.plot(cou1, "*", color='red', label = 'Party 1')  # Party_1をプロット
166     plt.plot(cou2, "x", color='green', label = 'Party 2')  # Party_2をプロット
167     # plt.legend()
168     plt.xlabel("Time (steps)")
169     plt.ylabel("Numbers of nodes in each party")
170     print('Drawing nou: {0}'.format(e))
171 # draw_count関数の終わり
172 
173 # パラメーター入力関数
174 # input_trust Realiablity levelとCreadit level, 近似範囲の入力
175 def input_trust():
```

```
    rev0 = float(input("Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
    while (rev0< 0) or (rev0 > 1):
        rev0 = float(input("入力が範囲を越えています。再度Party_1のActive nodesのメッセージの真偽性（realiability）を入力してください。（0-1)）: "))
    rev1 = float(input("Party_1のActive nodesの同じpartyのメンバーに対する信憑性（Credibility）を入力してください。（0-5)）: "))
    while (rev1 < 0) or (rev1 > 5):
        rev1 = float(input("入力が範囲を越えています。再度Party_1のActive nodesの同じpartyのメンバーに対する信用性（Credibility）を入力してください。（0-5)）: "))
    rev2 = float(input("Active nodesの間の影響度(R)を入力してください。（0-5)）: "))
    while (rev2 < 0) or (rev2 > 5):
        rev2 = float(input("入力が範囲を越えています。再度Active nodesの間の影響度(R)を入力してください。（0-5)）: "))
    return rev0,rev1,rev2
# input_trustの終わり

###############
# メインループ
###############
if __name__ == "__main__":
    # Party_1のActive nodesの[Active nodesの信頼性]の設定
    realiable_level, creadit_level, R = input_trust()
    trust_level =realiable_level * creadit_level
    r_ini = R   # 入力されたRを格納
    random.seed(SEED)  # 乱数の初期化
    cm = plt.get_cmap(CMAP)

    for et in range(EX):
        # Party_0のActive nodesの生成
        a = [Node(0) for i in range(N)]
    # Party_1のActive nodesを設定、座業を原点からずらす
        a[0].status = 0
        a[0].x = -2
        a[0].y = -2
    # Party 1をP個
        for p in range(P):
            a[p].status = 0
    # Party 1をM個
        for m in range(M):
            a[m].status = 1
    # Party 2をQ個
        for n in range(Q):
            a[n].status = 2

    # グラフデータの初期化
    # Party_0のデータ
        xlist0 = []
        ylist0 = []
    # Party_1のデータ
        xlist1 = []
        ylist1 = []
    # Party_2のデータ
        xlist2 = []
        ylist2 = []
    # ノード数
        sta0_sum = []   #Party 0のノード数の結果
        sta1_sum = []   #Party 1のノード数の結果
        sta2_sum = []   #Party 2のノード数の結果

```

```
    # 反転回数
        changelist = []
    # Active nodesシミュレーション
        for t in range(TIMELIMIT):
            calcn(a)  # 次時刻の状態を計算
            count0 = len(xlist0)
            count1 = len(xlist1)
            count2 = len(xlist2)
            sum_change = count1 - M
            changelist.append(sum_change)
            sum_change_per = sum_change/N *100
            draw_nodes(xlist0, ylist0, xlist1, ylist1,xlist2, ylist2,realiable_level,
creadit_level,t,count0, count1, count2,sum_change_per,et, R)
            # 描画データのクリア
            xlist0.clear()
            ylist0.clear()
            xlist1.clear()
            ylist1.clear()
            xlist2.clear()
            ylist2.clear()
            # 各ノードの総数
            sta0_sum.append(count0)   #Party 0のノード数の結果
            sta1_sum.append(count1)    #Party 1のノード数の結果
            sta2_sum.append(count2)    #Party 2のノード数の結果
        draw_count(realiable_level, creadit_level,sta0_sum, sta1_sum, sta2_sum, et,
R, N)
    #plt.legend(['Party 0','Party 1','Party 2'])
        plt.show() ####### 個別表示を標示

    print('\n========== The End of program ===========')

#  メインループの終わり

```

- ４.６.２インフルエンサーの導入　同数の集団に対する第３のノードの R の影響で第３集団がインフルエンサーである場合のノードソースコード

```
# -*- coding: utf-8 -*-
'''
=== Subject
*モチベーションのシミレーター(Active nodes)の導入
sim_202xxxxx.py

====履歴
*2021/11/20 勢力図の追加
*2021/09/05 P3ノードの導入と改変
*2021/08/09 Semi-logでの近似(First-order function approximation by least squares method)
*2021/06/01 Active nodesを導入
*2021/02/10 履歴を入れる
*2021/01/11 M_q の極性反転を入れる
*2020/12/31 Qの最大値、初期値の別での入力化、ランダムのスイッチを導入
*2020/11/01 3因子モデルに変更
*2020/11/14 メッセージの情報変容を導入


====利用論文
*2021/08/01 CCS Es2021 abstract


====利用について
コメントアウトすれば機能がする内容は #XXXXXX
で示してあります

'''

# モジュールのインポート
import random
import numpy as np
import matplotlib.pyplot as plt

# グローバル変数
# N = 100          # Active nodesの総個数
# M = int(N/2)     # Party 1に設定 ここで半数に設定
M = 50            # Party 1に設定
P = 50            # Party 0に設定
Q = 1            # Party 2に設定、インフルエンサー

N = M + P + Q    # Active nodesの総個

Rin = 2          # インフルエンサー
trust_level_in = 2 # インフルエンサーの信憑性


TIMELIMIT = 50  # シミュレーション打ち切り時刻
SEED = 65535     # 乱数の種
# SEED = 7
# R = 0.2           # 近隣を規定する数値
# trust_level = 1.0     # Party_1のActive nodesの初期信頼性の強さ
AREA = 25         # Agentの動き観察用描画範囲
#PAUSE_TIME = 0.001       # 描画間隔
PAUSE_TIME = 1     # 描画間隔
EX = 10           # 実験回数
STEP = 0.001      # Rの変化のステップ数
CMAP = "hsv"      # 複合グラフの色空間
#
```

```
# クラス定義
# Nodeクラス
class Node:
    """Active nodesを表現するクラスの定義"""
    def __init__(self, sta):  # コンストラクタ
        self.status = sta
        self.x = (random.random() - 0.5) * AREA  # x座標の初期値
        self.y = (random.random() - 0.5) * AREA  # y座標の初期値
    def calcnext(self):  # 次時刻の状態の計算
        if self.status == 0:
            self.sta0()  # Party_0の計算
        elif self.status == 1:
            self.sta1()  # Party_1の計算
        elif self.status == 2:
            self.sta2()  # Party_2の計算(第3勢力)
        else:  # 合致するPartyがない
            print("ERROR Partyがありません\n")
    def sta0(self):  # Party_0の計算メソッド
        # Party_1のActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 0:
                c0x = self.x
                c0y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c0x - ax) * (c0x - ax) + (c0y - ay) * (c0y - ay)) < R:
                    if a[i].status == 1:    # 隣接してParty_1のActive nodesがいる
                        self.status = 1  # Party_1に変身
                    elif a[i].status == 2:  # 隣接してParty_2のActive nodesがいる
                        self.status = 2  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level

    def sta1(self):  # Party_1の計算メソッド
        # 他のPartyのActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 1:
                c1x = self.x
                c1y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c1x - ax) * (c1x - ax) + (c1y - ay) * (c1y - ay)) < R:
                    if a[i].status == 0:    # 隣接してParty_0のActive nodesがいる
                        self.status = 0  # Party_0に変身
                    elif a[i].status == 2:  # 隣接してParty_2のActive nodesがいる
                        self.status = 2  # Party_2に変身
        else:   # 位置の更新
            self.x += (random.random() - 0.5) * trust_level
            self.y += (random.random() - 0.5) * trust_level

    def sta2(self):  # Party_2の計算メソッド
        # 他のPartyのActive nodesとの距離を調べる
        for i in range(len(a)):
            if a[i].status != 2:
                c2x = self.x
                c2y = self.y
                ax = a[i].x
                ay = a[i].y
                if ((c2x - ax) * (c2x - ax) + (c2y - ay) * (c2y - ay)) < Rin:
```

```
                    if a[i].status == 0:    # 隣接してParty_0のActive nodesがいる
                        self.status = 0  # Party_0に変身
                    elif a[i].status == 1:  # 隣接してParty_1のActive nodesがいる
                        self.status = 1  # Party_2に変身
            else:   # 位置の更新
                self.x += (random.random() - 0.5) * trust_level * trust_level_in
                self.y += (random.random() - 0.5) * trust_level * trust_level_in

    def putstate(self):  # 状態の出力
        print(self.status, self.x, self.y)
# Nodeクラスの定義の終わり
# 関数の定義
# calcn()関数
def calcn(a):
    """次時刻の状態を計算"""
    for i in range(len(a)):
        a[i].calcnext()
        # a[i].putstate()

        # グラフデータに現在位置を追加
        if a[i].status == 0:
            xlist0.append(a[i].x)
            ylist0.append(a[i].y)
        elif a[i].status == 1:
            xlist1.append(a[i].x)
            ylist1.append(a[i].y)
        elif a[i].status == 2:
            xlist2.append(a[i].x)
            ylist2.append(a[i].y)
# calcn()関数の終わり
#
# 描画関係の関数
# draw_nodes: Active nodeの動きを観察
def draw_nodes(xli0, yli0, xli1, yli1,xli2, yli2, reali, cread, ti, cou0,
cou1, cou2, sum_ch,exn, ran):
    plt.clf()

    fig, axes = plt.subplots(2,1, gridspec_kw=dict(height_ratios=[10,1]))
                        # メインを設定
    ax1 = axes[0]
    ax2 = axes[1]

    ax1.set_title('Realiablity level = {0}%,  Creadit level = {1}\n Time =
{2}\nParty 0 has {3} members, Party 1 has {4}, members Party 2 has {5}
members\nSum of change = {6}%\nNumber of Trial = {7}   Range R =
{8}'.format(reali * 100, cread, ti, cou0, cou1, cou2,sum_ch, exn+1, ran))
    ax1.axis([AREA*(-1), AREA, AREA*(-1), AREA])   # 描画領域の設定(暫定)
    ax1.plot(xli0, yli0, "o", color='blue', label = 'Party 0')  # Party_0をプロ
ット
    ax1.plot(xli1, yli1, "*", color='red', label = 'Party 1')  # Party_1をプロ
ット
    ax1.plot(xli2, yli2, "x", color='green', label = 'Party 2')  # Party_2をプ
ロット
    ax1.grid()
    ax1.legend()
    ax1.set_xlabel("Diffusion distance(x)")
    ax1.set_ylabel("Diffusion distance(y)")

    ax2.barh(0.8, [cou0, cou1, cou2], color = ['blue','red','green'])

```

```
    plt.pause(PAUSE_TIME)
# draw_nodes関数の終わり

# draw_count: Active nodeの動きを観察
def draw_count(reali, cread, cou0, cou1, cou2,e,ran, num):
    plt.title('Realiablity level = {0}%, Creadit level = {1}\nRange R = {2}, No. of Ex = {3}, Total Nodes ={4}'.format(reali * 100, cread,ran, e+1, num))
    plt.grid()
    plt.plot(cou0, "o", color='blue', label = 'Party 0')  # Party_0をプロット
    plt.plot(cou1, "*", color='red', label = 'Party 1')  # Party_1をプロット
    plt.plot(cou2, "x", color='green', label = 'Party 2')  # Party_2をプロット
    # plt.legend()
    plt.xlabel("Time (steps)")
    plt.ylabel("Numbers of nodes in each party")
    print('Drawing nou: {0}'.format(e))
# draw_count関数の終わり

# パラメーター入力関数
# input_trust Realiablity levelとCreadit level, 近似範囲の入力
def input_trust():
    rev0 = float(input("Party_1のActive nodesのメッセージの真偽性(realiability)を入力してください。(0-1)): "))
    while (rev0< 0) or (rev0 > 1):
        rev0 = float(input("入力が範囲を越えています。再度Party_1のActive nodesのメッセージの真偽性(realiability)を入力してください。(0-1)): "))
    rev1 = float(input("Party_1のActive nodesの同じpartyのメンバーに対する信憑性(Credibility)を入力してください。(0-5)): "))
    while (rev1 < 0) or (rev1 > 5):
        rev1 = float(input("入力が範囲を越えています。再度Party_1のActive nodesの同じpartyのメンバーに対する信用性(Credibility)を入力してください。(0-5)): "))
    rev2 = float(input("Active nodesの間の影響度(R)を入力してください。(0-5)): "))
    while (rev2 < 0) or (rev2 > 5):
        rev2 = float(input("入力が範囲を越えています。再度Active nodesの間の影響度(R)を入力してください。(0-5)): "))
    return rev0,rev1,rev2
# input_trustの終わり

###############
# メインループ
###############
if __name__ == "__main__":
    # Party_1のActive nodesの[Active nodesの信頼性]の設定
    realiable_level, creadit_level, R = input_trust()
    trust_level =realiable_level * creadit_level
    r_ini = R   # 入力されたRを格納
    random.seed(SEED)  # 乱数の初期化
    cm = plt.get_cmap(CMAP)

    for et in range(EX):
        # Party_0のActive nodesの生成
        a = [Node(0) for i in range(N)]
    # Party_1のActive nodesを設定、座業を原点からずらす
        a[0].status = 0
        a[0].x = R
        a[0].y = R
    # Party 1をP個
        for p in range(P):
            a[p].status = 0

    # Party 1をM個
```

```
        for m in range(M):
            a[m].status = 1
    # Party 2をQ個
        for n in range(Q):
            a[n].status = 2

    # グラフデータの初期化
    # Party_0のデータ
        xlist0 = []
        ylist0 = []
    # Party_1のデータ
        xlist1 = []
        ylist1 = []
    # Party_2のデータ
        xlist2 = []
        ylist2 = []
    # ノード数
        sta0_sum = []   #Party 0のノード数の結果
        sta1_sum = []   #Party 1のノード数の結果
        sta2_sum = []   #Party 2のノード数の結果

    # 反転回数
        changelist = []
    # Active nodesシミュレーション
        for t in range(TIMELIMIT):
            calcn(a)  # 次時刻の状態を計算
            count0 = len(xlist0)
            count1 = len(xlist1)
            count2 = len(xlist2)
            sum_change = count1 - M
            changelist.append(sum_change)
            sum_change_per = sum_change/N *100
            draw_nodes(xlist0, ylist0, xlist1, ylist1,xlist2,
ylist2,realiable_level, creadit_level,t,count0, count1,
count2,sum_change_per,et, R)
            # 描画データのクリア
            xlist0.clear()
            ylist0.clear()
            xlist1.clear()
            ylist1.clear()
            xlist2.clear()
            ylist2.clear()
            # 各ノードの総数
            sta0_sum.append(count0)   #Party 0のノード数の結果
            sta1_sum.append(count1)    #Party 1のノード数の結果
            sta2_sum.append(count2)    #Party 2のノード数の結果
        draw_count(realiable_level, creadit_level,sta0_sum, sta1_sum,
sta2_sum, et, R, N)
    #plt.legend(['Party 0','Party 1','Party 2'])
            #plt.show() ####### 個別表示を標示
        plt.show() ####### 連続表示を標示
    print('\n========== The End of program ===========')

# メインループの終わり

```

付録３：第４章のシミュレーションの動画結果

以下のファイルに収納。

- V1 2021-06-24 00-53-54.mp4
- V2 2021-06-24 00-56-16.mp4
- V3 2021-06-24 01-00-00.mp4
- V4 2021-06-24 01-03-20.mp4
- V5 2021-06-24 01-03-23.mp4
- V6 2021-06-24 01-07-08.mp4
- 454510R1influ.mov
- 50501R2Influe.mov
- 50501R2.mov
- 50501R0_1.mov