UNIVERSIDADE
LUSÓFONA

Modelamento multi-agente de interações em grupos sociais

-

Relatório Final

Nome do Aluno: Paulo Guilherme Pires de Jesus Pinto

Nome do Orientador: Manuel Marques Pita

Trabalho Final de Curso | LEI | 9/09/2022

www.ulusofona.pt

Direitos de cópia

Resumo

O Trabalho Final de Curso “Modelamento multi-agente de interações em grupos sociais” tem como objetivo desenvolver um modelo baseado em agentes, de forma a que se consiga estudar como variáveis individuais (como os enviesamentos cognitivos) afetam variáveis coletivas, e como as normas sociais influenciam comportamentos intra-grupo e inter-grupo através de simulações.

Variáveis individuais e normas sociais poderão, em conjunto, produzir comportamentos coletivos similares aos observados em situações reais. No caso de poder replicar comportamentos coletivos, o próximo passo será então entender quais são os mecanismos de ação das diferentes variáveis, e como tal informação poderá ser utilizada para promover diálogos online com maior qualidade.

Mais especificamente, o contexto de estudo é definido por atividades de debate de grupo online, nos quais os participantes são anónimos. Para tal, foi necessário recorrer e analisar literatura clássica de sistemas multi agente no contexto de redes sociais, de forma a que a construção do modelo seja válida. O modelamento será realizado através de inferências feitas a partir de dados existentes de debates de grupo. Todos os dados tratados no âmbito deste projeto, cedidos pelo Prof. Manuel Pita, são autênticos e foram recolhidos em escolas Portuguesas. Este TFC foi desenvolvido em NetLogo e Python, dando uso à técnica de Modelamento Multi-Agente (ABS), que consiste na criação e aperfeiçoamento de um modelo composto por agentes individuais, e posterior análise dos comportamentos emergentes.

Este relatório final tem como objetivo apresentar e explicar o modelo desenvolvido e demonstrar o trabalho que foi dedicado à realização do mesmo, para tal, ao longo deste relatório é apresentada a base científica e uma descrição detalhada do modelo e da sua implementação em NetLogo, conjuntamente com a sua análise em Python.

Abstract

The Final Project "Multi-agent Modelling of interactions in social groups" aims to develop a Multi-Agent System (MAS) to study how social norms influence intragroup and intergroup behaviours through the use of simulations. This Project requires studying Social Network Multi-Agent literature throughout its development, as it's oriented towards the possibility of being published in a scientific journal.

Individual variables and social norms may, together, produce collective behaviours similar to those observed in real situations. In the event that collective behaviour can be replicated, the next step will then be to understand the mechanisms of action of the different variables, and how such information can be used to promote higher quality online dialogues.

More specifically, the study context is defined by online group discussion activities, in which participants are anonymous. To this end, it was necessary to review and analyse classical literature on multi-agent systems in the context of social networks, so that the construction of the model is valid. Modelling will be carried out through inferences made from existing data from group discussions. All data processed within the scope of this project, provided by Prof. Manuel Pita, are authentic and were collected in Portuguese schools. This TFC was developed in NetLogo and Python, using the Multi-Agent Modelling (ABS) technique and subsequent analysis of emerging behaviours.

This final report aims to present and explain the model developed and demonstrate the work that was dedicated to its realization, for this purpose, throughout this report, the scientific basis and a detailed description of the model and its implementation in NetLogo and its analytics in Python.

Índice

Lista de Figuras

1 Identificação do Problema

Um pressuposto frequentemente associado ao comportamento dos jovens de um grupo social amplo, como por exemplo um país, é que os seus comportamentos são, em média, similares. Em estudos recentes de comportamento de jovens em situação de conversa social online, a equipa de investigação do projeto Debaqi, coordenado pelo orientador deste projeto, identificou comportamentos de grupo muito diferentes entre si. Estes comportamentos são consistentes com diferenças profundas de grupo social.

São muitas as *micro-variables* que definem um grupo social. Num micro-contexto específico, como as atividades de conversa online, não sabemos a priori que variáveis regulam o comportamento dos jovens. Neste projeto vamos considerar o modelamento multiagente de entidades que consideram as seguintes variáveis, com o objetivo de tentar reproduzir comportamentos coletivos observados em dados recolhidos durante 2019 e 2020:

1. Grau de conhecimento
2. Propensão a contribuir com intervenções
3. Propensão a responder a intervenções de outras pessoas
4. Foco (Explorar vs. Aprofundar)
5. Compromisso com a atividade de conversa
6. Propensão a adesão/rejeição das opiniões de outros sem argumentar

Nestes dados foram observadas certas dinâmicas intra-grupo distintas entre os grupos sociais ao qual os alunos integravam. Os alunos de classes trabalhadoras foram mais propensos a sabotar a conversa através de uma desvalorização do conhecimento de outros colegas. Os alunos de classe média conseguiram interagir mais entre si e consequentemente houve mais troca de informação, também tiveram um comportamento particular dicotómico de seguidor/não seguidor, em que alunos estabeleciam quase que "cliques" mediante a sua posição num determinado assunto. Quanto aos alunos de classe alta observou-se menos diálogo e pouca partilha de informação, em troca, os alunos adotaram uma atitude muito individualista, optando por intervir quase em monólogo: "despejavam" o seu conhecimento na matéria e não prolongavam o diálogo nem trocavam

de tópico facilmente. Este grupo, ao contrário dos grupos de classe trabalhadora, não aceitava sabotagem na conversa, entenderam que aquele devia ser um espaço sério e reduziram as suas interações com os alunos que não seguiam essa norma dentro da plataforma.

Estes resultados preliminares demonstram, portanto, que não existe um grande grupo social de jovens portugueses na perspetiva da conversa social, mas sim vários. Isto significa que qualquer intervenção educativa que vise melhorar as aptidões de conversa social dos jovens deve considerar estas diferenças. Neste trabalho de final de curso o problema central consiste em usar grandes volumes de dados para informar a natureza das variáveis individuais e coletivas a incluir num modelo mínimo que consiga replicar os comportamentos observados, e a partir desse modelo estudar o que poderá acontecer em diversos cenários, incluindo aqueles onde existem intervenções.

2 Viabilidade e Pertinência

O projeto visa criar uma primeira versão de um modelo multiagente através do qual consigamos reproduzir comportamentos observados num conjunto de dados relativamente reduzido, o qual será testado posteriormente em novos dados a ser adquiridos durante o primeiro e segundo períodos letivos de 2022. O projeto, numa versão mais avançada, poderá ser parte de um dos produtos científicos de um projeto financiado pela Fundação para a Ciência e a Tecnologia (FCT) em parceria com o Ministério da Educação referência DSAIPA/DS/0102/2019.

É importante para o Ministério da Educação identificar uma estratégia para melhorar as interações sociais dos alunos online, promovendo espírito crítico, literacia digital e diálogo saudável, mas como os dados recolhidos no programa Debaqi evidenciam, não há uma estratégia universal que funcione para todos, porque grupos sociais diferentes precisam de incentivos diferentes, algo que este projeto visa analisar com mais profundidade. Vão ser identificadas variáveis críticas que alterem o comportamento dos grupos e vão ser trabalhadas várias hipóteses como por exemplo "aumentando o grau do conhecimento dos alunos nota-se um impacto positivo no sistema".

O projeto é relevante porque contribui positivamente a resolver um problema atual, nomeadamente a participação dos jovens numa sociedade plural e democrática que valoriza a verdade e as interações baseadas em evidências.

3 Background

Resultados empíricos de estudos recentes no tema de redes sociais estão cada vez mais a desafiar a até agora dominante visão estruturalista (*top-down*), em sociologia. Estruturalismo social propõe que as dinâmicas sociais coletivas emergem duma estrutura social subjacente dos membros que interagem. Por outro lado, abordagens micro-sociológicas como o Interacionismo Simbólico defendem que estruturas sociais emergentes são o resultado de interações elementares entre agentes. Contudo, visões teóricas mais recentes sugerem que é uma combinação de processos *bottom-up* e *top-down* que constroem a realidade social: o nível micro leva à emergência do macro e, em torno, o macro afeta as interações locais. O mediador fulcral de emergência social é então o ato de comunicação em si (Jan Fuhse, 2015).

Um crescente número de métodos sociológicos propostos ao longo dos últimos vinte anos integram esta perspetiva teórica de Redes de visão de comunicação. Um destes métodos foca em como as pessoas mudam o assunto durante uma conversa (Okamoto and Smith-Lovin, 2008). Outro analisa a auto-organização *bottom-up* de coerência conversacional (Barzilay and Lapata, 2008).

O projeto para o qual este TFC contribui recolheu uma grande amostra de debates de alunos do ensino secundário. Há informação acerca da idade, género e do estrato social predominante da escola dos participantes. Estes dados possibilitam gerar probabilidades empíricas relativas ao comportamento de estudantes em debates online. Além disso, a análise dos debates revelou diferentes ritmos de troca de tópico nas conversas.

A hipótese em consideração é que pelo menos dois fatores determinam as diferenças observadas: estrato social e género. Estrato social determina comportamentos conversacionais ao haver diferentes normas no que toca à abordagens de diversos temas. Por exemplo: alunos em escolas de classe alta estão mais propensos a discutir sobre assuntos como as alterações climáticas e equidade de género e muitas vezes iniciam longos monólogos porque – na sua norma social – é esperado que o indivíduo demonstre quanto sabe acerca de determinado tópico. Neste estrato social, não foram observadas diferenças significativas entre géneros no que toca à participação.

Reciprocamente, para os estudantes da classe trabalhadora, a norma é a disrupção perante narrativas que demonstrem conhecimento. Isto é feito trocando o assunto para algo banal, muitas vezes através de piadas ou até com a ridicularização do aluno que quer participar duma forma mais intelectual na conversa. Neste estrato social há diferenças no comportamento entre os dois géneros: em regra geral, as raparigas procuram participar na conversa, ouvir os outros e contribuir com ideias racionais, enquanto que os rapazes valorizam mais a disrupção.

Este trabalho começou com o propósito de replicar um modelo existente, produzido em (Schlaile, 2021), que parte à descoberta de uma nova maneira de entender a difusão e assimilação de conhecimento em redes de inovação, com recurso a *Agent Based Modelling* (ABM). O modelo proposto pelos autores consiste numa rede de redes de conhecimento (abordagem *Network of Networks*). Este modelo foi escolhido porque tem em conta vários aspetos da interação humana considerados relevantes para modelar grupos de pessoas, num contexto de interação online.

Para fazer a sistematização de como o conhecimento é propagado, os autores baseiam-se noutros artigos científicos que provam que para alguém obter um novo "pedaço" de conhecimento, precisa primeiro de ter uma base ou um conhecimento um mínimo similar para conseguir conciliar a nova informação. Para exemplificar este fenómeno podemos refletir no seguinte: tópicos de Ciência de Dados são mais facilmente assimilados por matemáticos dados à estatística e programadores do que a alunos de cinema (conhecimento similar); operar um câmara de filmagem é mais fácil para alguém que tenha experiência com uma câmara de fotografia (base de conhecimento).

O modelo desenvolvido representa o que um agente conhece na forma de uma rede de unidades de conhecimento (*knowledge units*, KU), representadas por uma lista binária. Se a similaridade entre duas KUs da mesma rede estiver acima de um certo limiar, dizem-se ligadas. Esta forma de ligar KUs acaba por formar aglomerados em determinados tópicos/domínios. Estas aglomerações são a base da mudança de política de atenção de um agente entre uma exploratória (explora novos aglomerados) ou uma que continue a especializar o conhecimento atual (parmanece focado no mesmo aglomerado). A modelação de modos de atenção considera que agentes em interação não estão num estado *tabula rasa*, têm dinâmicas internas de atenção.

Uma vez instanciados os diferentes agentes e que cada um possua a sua rede de conhecimento e dinâmica interna de atenção, podemos modelar as interações entre agentes. No modelo original (Schlaile, 2021) as interações entre agentes são meramente diádicas. Uma adaptação feita neste modelo foi transformar esta interação para que os agentes agora interajam com uma *message board* comum a todos, algo que melhor descreve a situação que estamos a estudar. Com esta alteração feita podemos modelar como as mensagens que aparecem na *board* (e quem as publica) alteram o modo de interação dos agentes.

Em suma, este TFC procura produzir um Modelo de Simulação Multi-Agente que nos permita explorar variações na emergência de realidade social, que é representada aqui pelas conversas de grupo orientadas a tarefas, no sentido em que há questões concretas sobre um tópico central.

Na fase de desenvolvimento seguinte ao TFC, serão incrementalmente adicionadas variáveis que descrevem tendências comportamentais individuais - como: fala, mudança de tópico, ser influenciado/a – e variáveis de grupo - como quanto é que o grupo de agentes valoriza o conhecimento num certo tópico vs sabotar a discussão. Depois desta fase o modelo continuará a ser aprimorado e revisto, tendo em vista a sua otimização para reproduzir os comportamentos coletivos observados nos dados disponíveis.

4 Benchmarking

Sendo este TFC de componente científica, ao invés dum *Benchmarking*, foi feita uma análise inicial de alguns documentos científicos que abordassem o assunto de ABM e as suas possíveis aplicações, com o intuito de apresentar o enquadramento cientifico do projeto.

Modelamento baseado em agentes é uma técnica de simulação de sistemas, que consiste em descrever um sistema a partir da perspetiva das unidades constituintes. Em ABM, os sistemas são regidos por vários agentes autónomos com comportamentos (por norma) simples, definidos a partir de um conjunto de regras. Sendo que ABM se baseia nesta descrição natural dum sistema, as suas aplicações podem tomar as mais variadas temáticas, desde a economia até às ciências sociais.

Neste tipo de simulação é normal ocorrerem fenómenos emergentes inesperados aquando a concepção do modelo. Um fenómeno emergente nada mais é do que um acontecimento ao nível do sistema, despoletado pelas interações individuais dos agentes.

O artigo (Srour, 2017) põe a questão: *Conseguirá ABM ensinar-nos algo sobre os fatores sociais ligados à corrupção?* Esta questão é estudada através dum modelo que representa trabalhadores de docas, com tendência a subornar contentores (objetos de corrupção). Os autores concluem que devido à interconexão de agentes, normas sociais que tentem acabar com a corrupção numa área do sistema apenas a deslocam para outra.

No artigo (Zhang, 2020) os investigadores implementaram um modelo de polarização para analisar como a conformidade e a rede de relações de alguém impacta a forma como interagem e leva a uma eventual polarização. Para confirmar os resultados implementaram um modelo multiagente com o método de Monte Carlo.

O artigo (Laskowski, 2010) procura entender as dinâmicas de conversas de grupo, nomeadamente quem participa e como. O investigador teve o cuidado de escolher para análise dados não influenciados pela existência da sua pesquisa.

5 Solução Desenvolvida

A solução desenvolvida é um modelo baseado em agentes e um conector em Python para análise de dados. O modelo em si é um simulação desenvolvida em Netlogo que possui agentes com valores iniciais distintos mas as mesmas restrições na forma em como interagem entre si, através de uma *board* central.

Ao iniciar a simulação, são gerados agentes com redes de conhecimento e uma *board* com o tópico inicial da conversa constante. Em cada *tick* (unidade temporal da simulação) os agentes definem a unidade de conhecimento que se vão focar a partir de um de dois mecanismos diferentes: um que prioriza desenvolver o conhecimento atual e portanto encontra uma unidade de conhecimento similar à que estava em foco, e outro que prioriza a exploração de conhecimento, encontrando uma dissimilar. Com a unidade de conhecimento em foco definida, os agentes vão então avaliar a compatibilidade da mesma com as que estão na *board* e, caso sejam compatíveis, ela é enviada para a *board*.

Durante o correr da simulação são guardados vários dados como quantos agentes estão a participar a cada *tick* e a sua distribuição por género, as unidades de conhecimento previamente presentes na *board* e até que agentes contribuíram e em que *tick*. Com estes dados organizados e interpretados, obtemos informação sobre a simulação, que permite refinar a solução, validar dados e ponderar os comportamentos implementados duma maneira informada.

O conector de Python permite correr várias simulações com parâmetros diferentes de forma automatizada, poupando muito tempo e trabalho manual. À medida que estas simulações são executadas é possível guardar dados em ficheiros .csv para mais tarde consultar ou até gerar gráficos sem ser necessário executar a simulação de novo.

Todo este funcionamento tem como objetivo replicar as dinâmicas de uma sala de conversa online e está explicado em maior detalhe na secção 8, que também inclui os detalhes da implementação do modelo.

Tecnologias utilizadas no desenvolvimento deste TFC:

- **NetLogo** – Ferramenta e linguagem de programação orientada para o desenvolvimento e análise de modelos. Tem funcionalidades que permitem gerir os agentes, o ambiente e as várias interações que acontecem dentro do sistema, que podem gerar comportamentos emergentes interessantes e/ou inesperados.
  Também vem equipado com uma interface gráfica munida de ferramentas de análise de aplicação rápida, como por exemplo:
  - mapa/ecrã para visualizar o comportamento do modelo no espaço
  - *sliders*, *switches* e *inputs* para modificar facilmente as variáveis globais
  - gráficos para fazer uma análise rápida à medida do tempo

Com a representação gráfica do NetLogo, é possível identificar comportamentos emergentes (comportamentos que os agentes individualmente não produzem, mas em grupo sim), como por exemplo o modelo "*Flocking*", da biblioteca de modelos do NetLogo. Este modelo ilustra bem o que são comportamentos emergentes, os agentes simplesmente procuram aproximar-se de outro agente ao pé de si e, eventualmente, é formado um bando de múltiplos agentes que se movimenta como um todo (este modelo é baseado no comportamento de alguns tipos de aves).

Cadeiras do curso relevantes para a elaboração do TFC:

- Fundamentos de Programação
- Linguagens de Programação I
- Linguagens de Programação II
- Matemática I
- Matemática II
- Ciência de Dados
- Inteligência Artificial

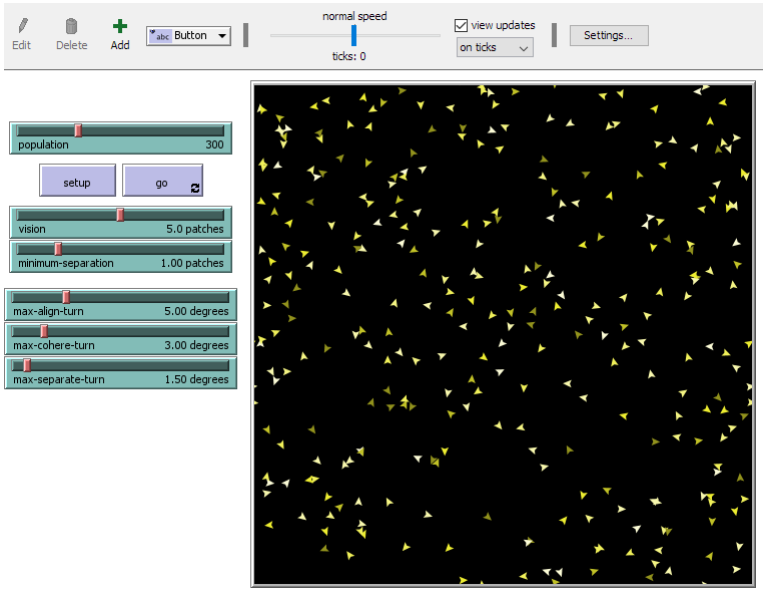


Figura 1 - Modelo ao início, agentes espalhados

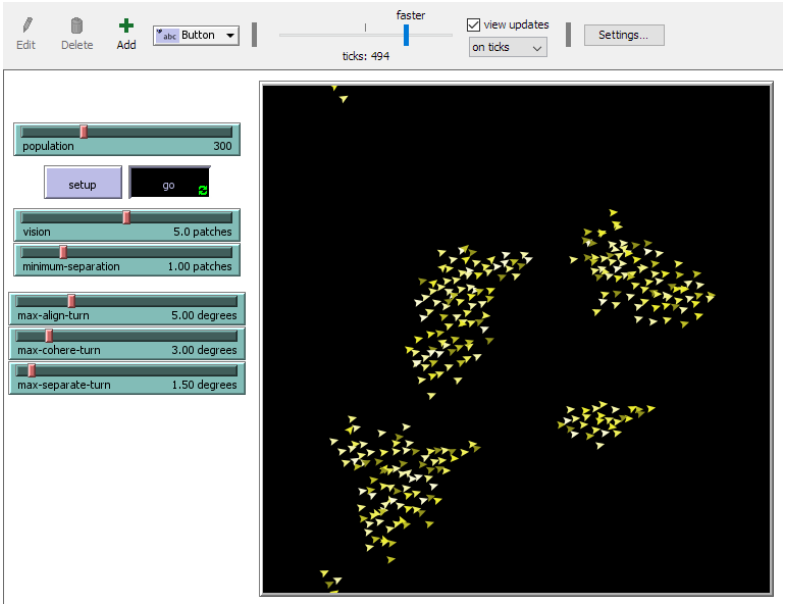


Figura 2 - Modelo em execução, agentes juntos (flocking)

- **Python (PyNetLogo)** - Linguagem de programação de alto nível para fazer a conexão com o modelo NetLogo e como ferramenta para análise dos dados provenientes das simulações, dando especial destaque às bibliotecas Pandas, Numpy e Matplotlib.

Foi estipulado numa fase inicial cumprir estes requisitos dando uso à biblioteca PyLogo, uma biblioteca open-source que foi criada porque foram identificadas funcionalidades que faltavam em NetLogo e que o Python disponibiliza (como dicionários e *sets*) e outras em que o NetLogo era ambíguo ou tinha sintaxe confusa, como é o caso do *if* e *while*, que têm duas maneiras diferentes de declarar a condição, no entanto esta biblioteca serve meramente como substituto do NetLogo, não cumprindo os requisitos definidos para este projeto.[1]

A biblioteca escolhida para fazer a "ponte" entre o modelo NetLogo e o Python foi a biblioteca PyNetLogo, que disponibiliza funcionalidades que tornam todo este processo muito indolor.[2] São exemplos dessas funcionalidades as seguintes:

- ***NetLogoLink***, *Constructor* - instancia um objeto com a conexão à aplicação NetLogo (com ou sem GUI) e retorna-o, podendo associá-lo assim a uma variável. As seguintes funções estão implementadas nesta classe
- ***load_model()***, importa o modelo situado no *path* indicado
- ***command()***, executa um comando como se fosse na linha de comandos do NetLogo
- ***report()***, executa um *reporter*, que é uma função que devolve algum tipo de dados
- ***repeat_report()***, igual ao *report()*, mas executa o número de vezes pretendidas. É bastante útil porque permite retornar múltiplas variáveis simultaneamente e criar um *DataFrame*, para guardar facilmente os dados da simulação.

[1] Abbott, R., & Lim, J. S. (2021). PyLogo: A Python Reimplementation of (Much of) NetLogo.

[2] Jaxa-Rozen & Kwakkel (2018). PyNetLogo: Linking NetLogo with Python, Journal of Artificial Societies and Social Simulation, 21

6 Resultados

Para análise de resultados foram selecionados 3 cénarios exemplo com as seguintes configurações:

Cenário	Taxa *exploit* feminina	Taxa *exploit* masculina	Chance Posting
1	0.1	0.1	1
2	0.9	0.9	1
3	0.5	0.5	1

Cenário 1 – fx=0,1 mx=0,1 pc=1

Compatibilidade

Neste cenário, evidencia-se que o comportamento predominante foi a incompatibilidade. Isto é devido ao comportamento '*explore'* - gerado pela baixa probabilidade do comportamento '*expoit'* - que causa com que os agentes se afastem do tópico, explorando novos temas.

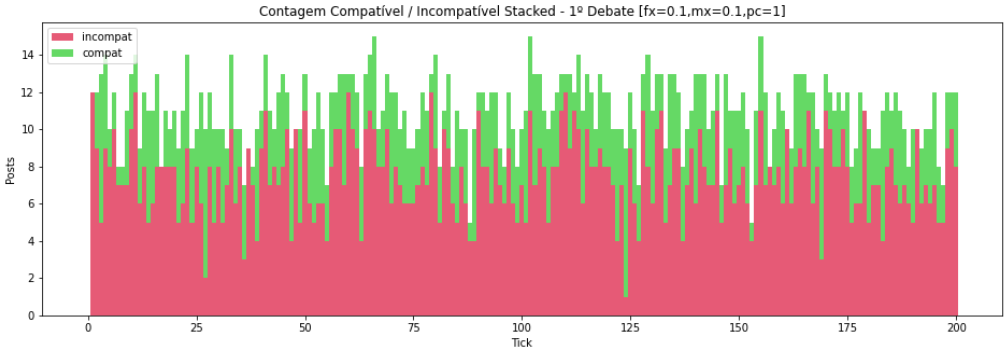


Figura 3 - Contagem *Posts* compatíveis e incompatíveis por *Tick*, no Debate 1 do Cenário 1

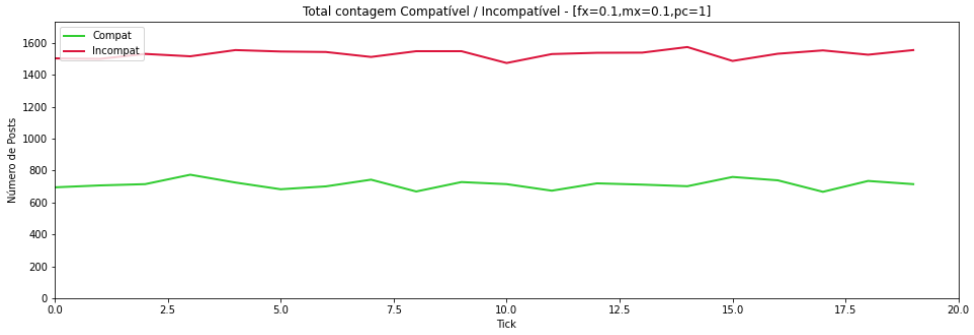


Figura 4 - Total de *Posts* compatíveis e incompatíveis por cada Debate do Cenário 1

Duração Conversas

A conversa pode-se manter *in-topic*, onde os bursts têm um grau de compatibilidade superior a 50% com a KU inicial, ou *out-of-topic*, onde não o têm. Neste cenário as conversas *in-topic* nunca ultrapassaram a duração de 2 ticks, durante 20 debates com 200 ticks cada.

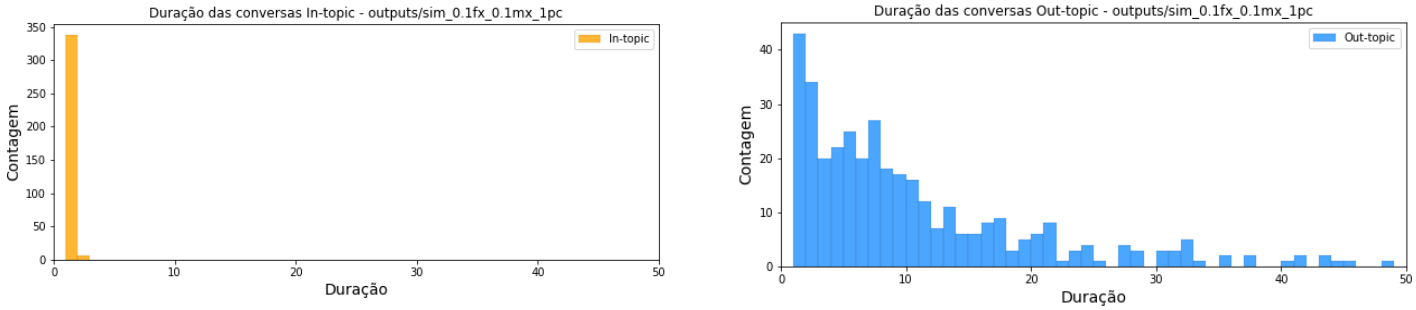


Figura 5 e 6 - Duração das conversas *in* e *out-of-topic* no Cenário 1, respetivamente

Contagem de *Posts* por *Burst*

Nesta métrica é avaliada a densidade das conversas, e neste cenário observa-se que houve muita participação, sendo a moda 12 *posts* por *burst*.

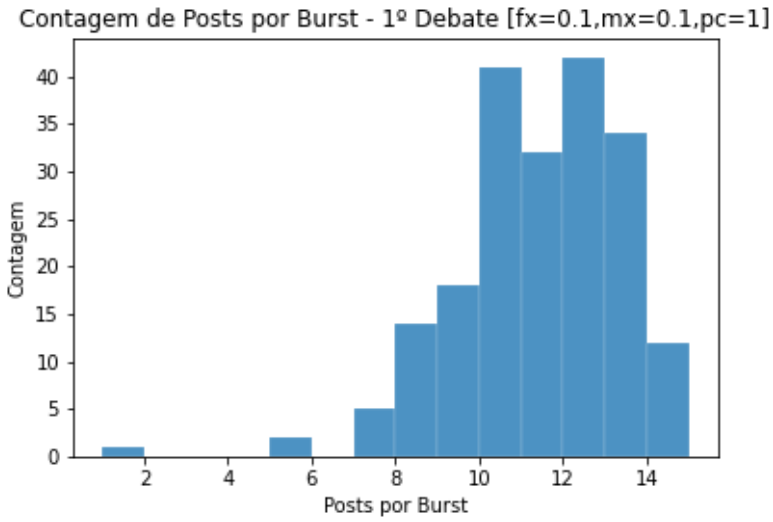


Figura 7 - Contagem de *Posts* por *Burst* Cenário 1

Cenário 2 – fx=0,9 mx=0,9 pc=1

Compatibilidade

Neste cenário há uma tendência muito maior para a compatibilidade, que gera um comportamento polarisante. A Norma de Atenção '*Exploit*' dominante neste cenário causa com que as KUs em foco mantenham uma baixa distância de Hamming com a KU inicial, isto é, uma compatibilidade fiável.

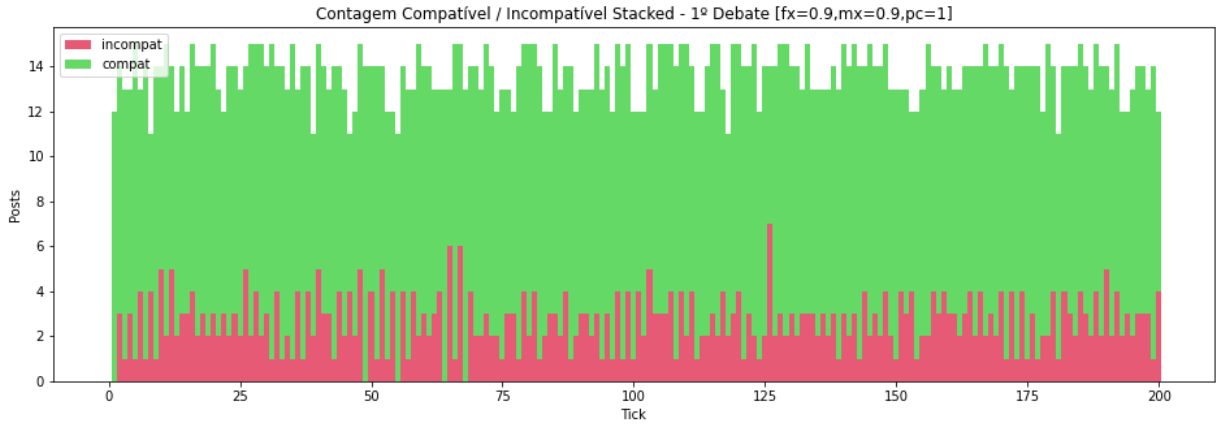


Figura 8 - Contagem *Posts* compatíveis e incompatíveis por *Tick*, no Debate 1 do Cenário 2

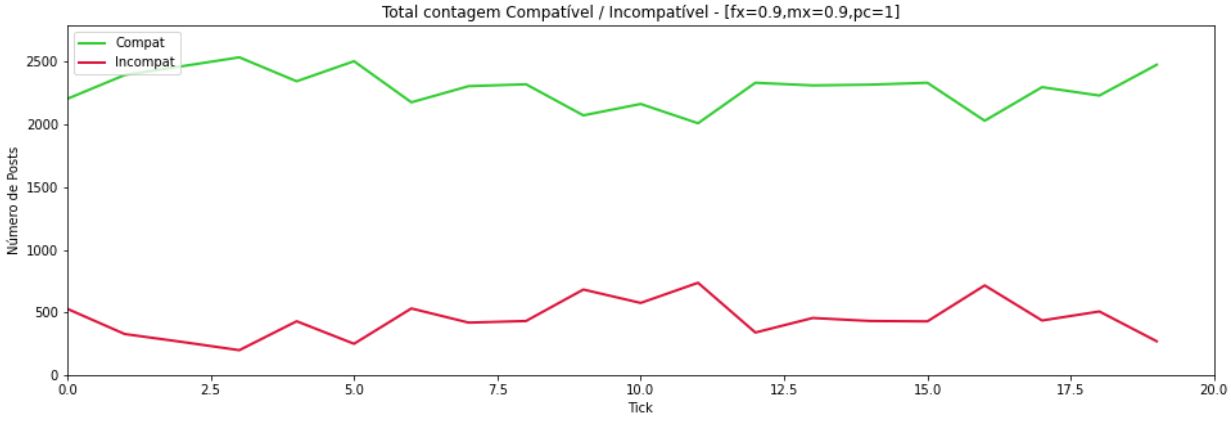
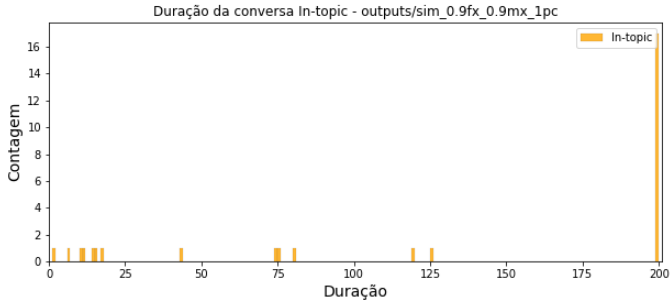


Figura 9 - Total de *Posts* compatíveis e incompatíveis por cada Debate do Cenário 2

Duração Conversas

Para este cenário, 16 dos debates mantiveram ininterruptamente durante a sua duração total (200 *ticks*) uma conversa *in-topic*.



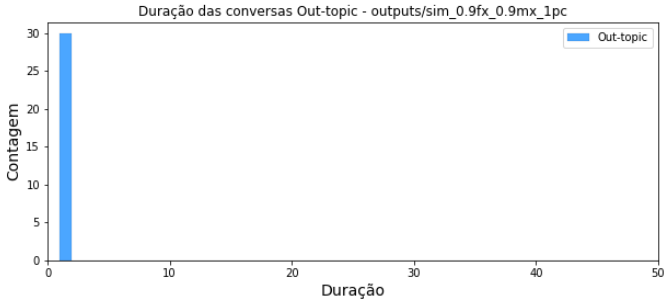


Figura 10 e 11 - Duração das conversas *in* e *out-of-topic* no Cenário 2, respetivamente

Contagem de *Posts* por *Burst*

A participação é demasiado acentuada neste cenário, todos os *bursts* (exceto 1) tiveram mais de 11 agentes a participar, no 1º Debate dos 20 simulados.

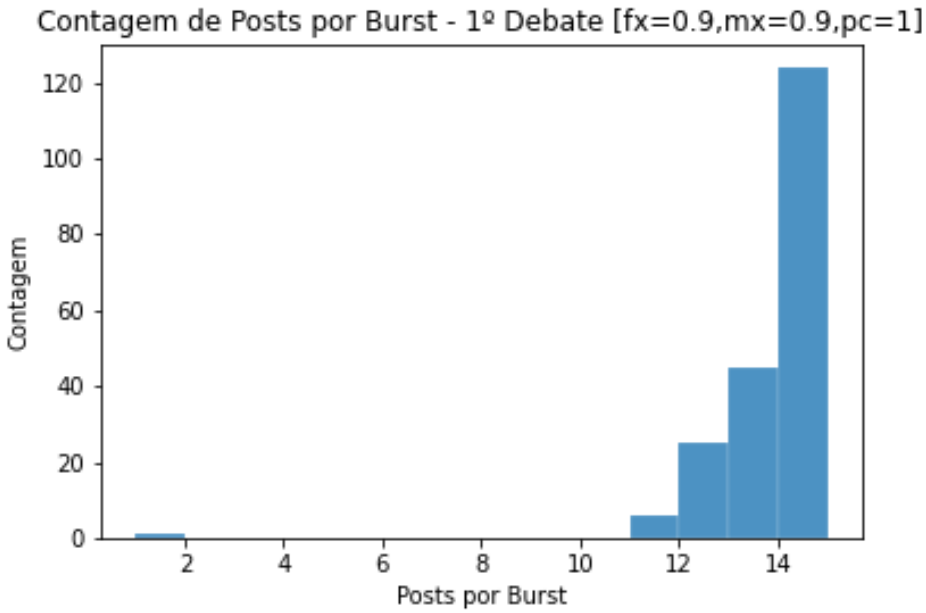


Figura 12 - Contagem de *Posts* por *Burst* Cenário 2

Cenário 3 – fx=0,5 mx=0,5 pc=1

Compatibilidade

Este cenário tem comportamentos dispersos porque a probabilidade do comportamento '*exploit'* é a mesma que a do '*explore'*, resultando numa distribuição igualitária dos agentes entre os dois comportamentos que acaba por prevenir que as KUs enviadas para a board tendam demasiado para a proximidade com a KU inicial ou para o seu afastamento dela.

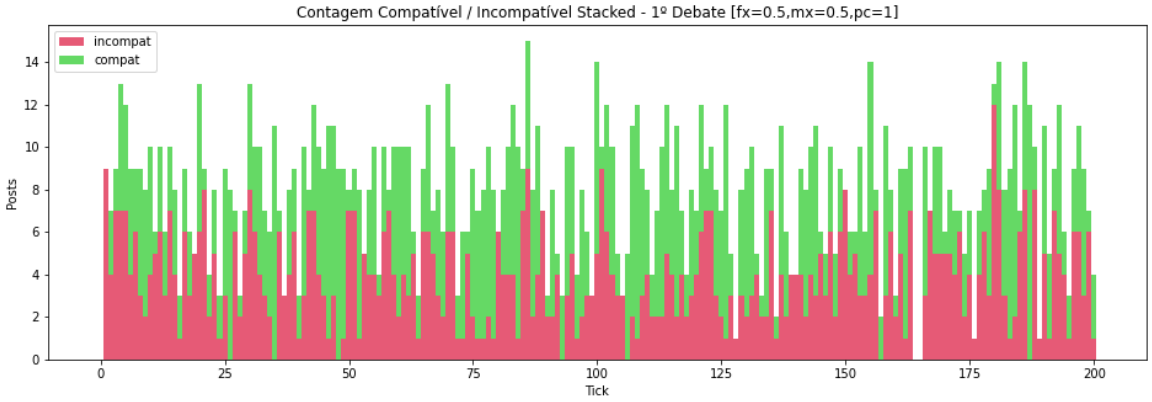


Figura 13 - Contagem *Posts* compatíveis e incompatíveis por *Tick*, no Debate 1 do Cenário 3

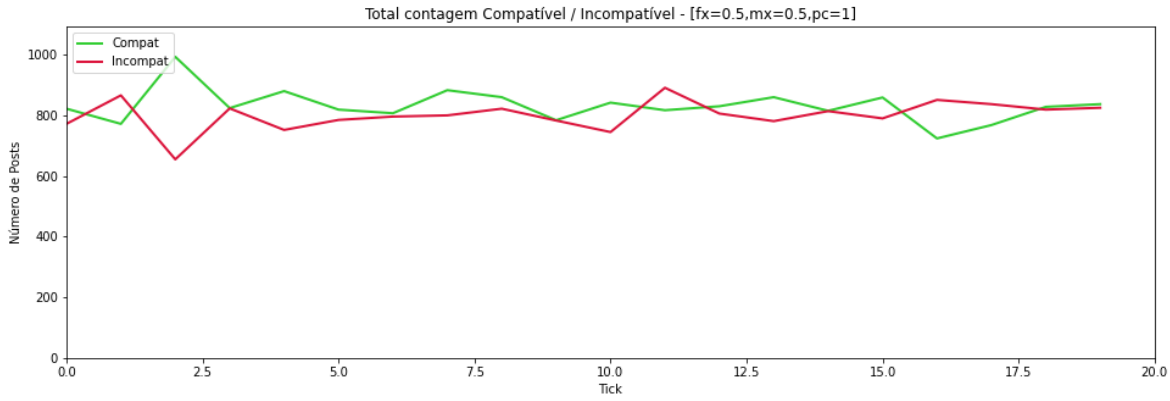


Figura 14 - Total de *Posts* compatíveis e incompatíveis por cada Debate do Cenário 3

Duração Conversas

Uma consequência adicional da distribuição de comportamentos neste cenário é que as conversas não permanecem muito tempo *in* ou *out-of-topic*, são poucos os casos em que há uma sequência de 10 ou mais *ticks* para cada uma das opções.

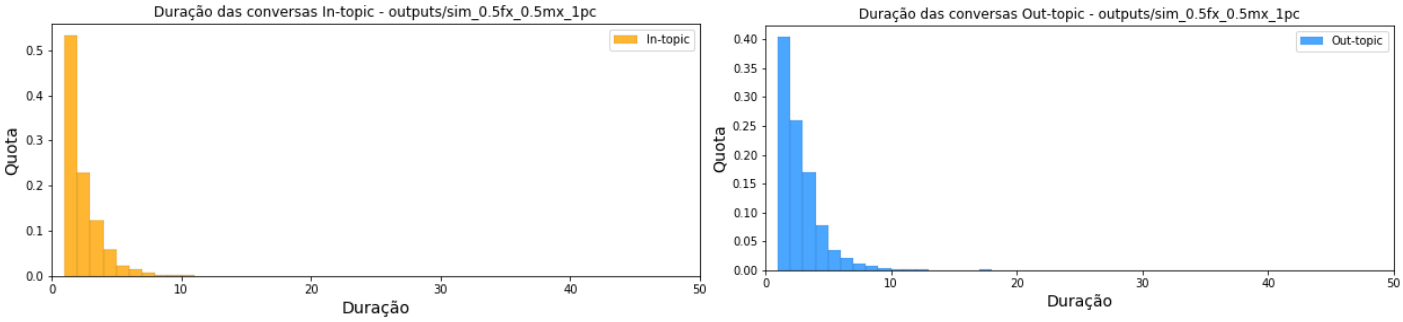


Figura 15 e 16 - Duração das conversas *in* e *out-of-topic* no Cenário 3, respetivamente

Contagem de Posts por Burst

A participação neste cenário é bastante dividida, com uma moda de 10 *posts* por burst no 1º Debate.

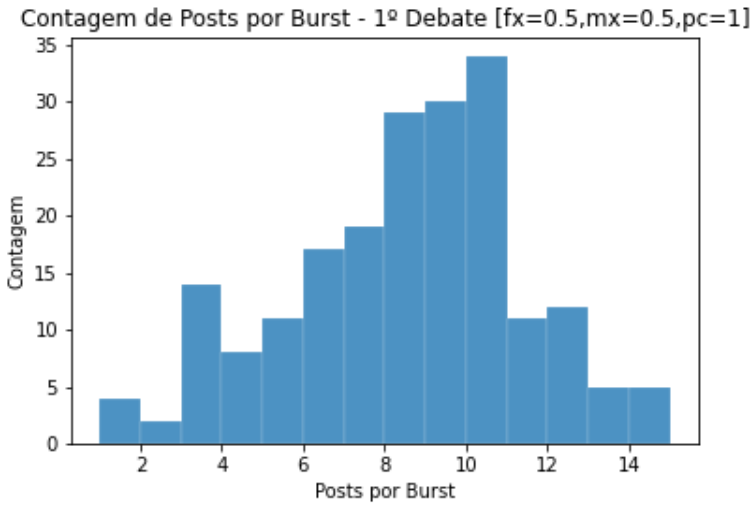


Figura 17 - Contagem de *Posts* por *Burst* Cenário 3

Estes cenários servem para exemplificar algumas das diferenças mais percetíveis quando se mudam variáveis da simulação. Também ajudam a demonstrar como se podem analisar dados desta simulação, com visualizações que transmitem informação muito díficil de definir/transmitir noutros meios como o texto.

7 Implementação

Na primeira entrega do projeto, foram implementadas em NetLogo componentes do artigo referido, tais como a inicialização dos agentes e o funcionamento das suas redes de conhecimento. Estas implementações continuam presentes na versão de entrega final e são complementadas por um novo tipo de agente - *board* - com o seu conjunto de ações e interações com o seu ambiente.

Com o modelo nesta fase foi possível começar a avaliar algumas métricas para analisar o comportamento dos agentes em diferentes cenários. A implementação por inteiro pode ser consultada como no repositório de github criado para esse efeito.

7.1 KNETs e KUs

Neste modelo, o tipo de agentes mais "rudimentar" possui redes de conhecimento (*knowledge networks* / KNET) compostas por unidades de conhecimento (*knowledge units* / KU). Cada KU é representada como um *array* de bits. A variabilidade do seu tamanho está implementada e é usado o tamanho de 16 bits (0 a 65535).

Para que todas as simulações tenham um ponto de partida relativamente similar, a *board* é inicializada com uma KU fixa (255, em binário) e a geração das KNETs dos agentes segue uma percentagem pré-definida de 40 KU compatíveis e 110 incompatíveis.

A função setup-turtles inicializa os agentes, que por si criam KUs, cada uma com um valor compreendido entre 0 e 2^*ku_len* (o tamanho estabelecido). Todas são inseridas na lista 'kus'.

```
set kus []
repeat ku_number [
  let this_ku random (2 ^ ku_len)

  ;; check if any turtle already has this value as its content
  while [member? this_ku kus]
  [
    set this_ku random (2 ^ ku_len)
  ]
  set kus insert-item length kus kus this_ku
]
```

Figura 18 - Criar KUs

As KUs são inicializadas e guardadas como decimais, portanto está implementada uma função para passar de decimal para lista de binários, '*decimal_to_binary'*.

7.2 Distância de Hamming e Compatibilidade

Distância de Hamming é a distância entre dois *arrays* de *bits*.

```
to-report normalized-hamming-distance [ bit_arr_1 bit_arr_2 ]
  report (length remove true (map [[?1 ?2] -> ?1 = ?2] bit_arr_1 bit_arr_2)) / ku_len
end
```

Figura 39 - Função para calcular a distância de Hamming e a compatibilidade, em NetLogo

A função *normalized-hamming-distance* recebe dois *arrays* de *bits* e aplica a operação lógica AND a todos os pares de elementos ao longo dos arrays. Estando esta operação feita, ficamos com um novo *array* de *bits*, em que o 1 representa os elementos que são iguais entre as duas listas, e o 0 representa os que são diferentes. O próximo passo é remover os 1's e contar o número de elementos que restam no *array*, que nos dará a distância de Hamming entre eles. Para comparar com um limiar de compatibilidade fixo, é preciso que esta métrica esteja normalizada (compreendida entre 0 e 1), para tal basta dividir pelo tamanho do array.

Tomemos por exemplo:

1. Listas de *bits* [0 1 1 0 0 1 0 0] e [1 1 1 1 0 1 1 0];
2. É aplicado o AND (com a função *map*) e obtemos o array [0 1 1 0 1 1 0 1];
3. Removemos os 1's e ficamos com [0 0 0], a distância de Hamming é 3;
4. Para normalizar dividimos 3 por 8, que resulta numa distância de Hamming normalizada de 0.375.

A compatibilidade entre duas KUs é o complementar da distância de Hamming normalizada a 1, como implementado na seguinte função.

```
to-report compatibility [ bit_arr_1 bit_arr_2 ]
  report 1 - (normalized-hamming-distance bit_arr_1 bit_arr_2)
end
```

Figura 20 4- Função para calcular a compatibilidade, em NetLogo

```
[1 1 0 0 0 1 0 0] and [1 1 0 0 1 1 0 0] are compatible - 0.875
[1 1 0 0 0 1 0 0] and [1 1 0 1 1 0 0 0] are compatible - 0.625
[1 1 0 0 1 1 0 0] and [1 1 0 1 1 0 0 0] are compatible - 0.75
```

Figura 51 - Cálculos de compatibilidade

7.3 Ligações

Por cada par único de KUs compatível, é criada uma ligação, que é guardada na variável de agente '*list_of_links*'.

Estas ligações são geradas pela função '*set_links*', que recebe um agente e itera por todas as combinações de duas KUs desse agente, verificando se são diferentes e compatíveis; se esta condição se aplicar e a ligação ainda não constar na lista de ligações, é criada e adicionada a uma lista temporária de ligações que, posteriormente, é adicionada ao '*list_of_links*' do agente.

```
to-report set_links [agent]
  ;; get copy of agent's kus
  let kus_list sort [kus] of agent

  let temp_list [list_of_links] of agent

  ;; for each pair of kus ( ku_a, ku_b )
  foreach kus_list [ ku_a ->
    let a decimal-to-binary ku_a

    foreach kus_list [ ku_b ->
      let b decimal-to-binary ku_b

      ;; are they different?
      if a != b [
        ;; are they more compatible than the compatibility threshold?
        if compatibility a b > c_threshold [
          ;; create tuple ( a, b )
          let tuple create_link ku_a ku_b agent

          ;; check for duplicates and add to temporary list
          if not member? tuple temp_list [
            type a type " and " type b type " are compatible - " print (compatibility a b)
            set temp_list insert-item length temp_list temp_list tuple
          ]
        ]
      ]
    ]
  ]
  ;; report temporary list, to add to agent
  report temp_list
end
```

Figura 226 - Função para encontrar as ligações e adicionar ao agente

```
to-report create_link [ a b agent ] ;; 2 KUs

  if b < a [
    let temp b
    set b a
    set a temp
  ] ;; a is always the smallest number

  ;; create tuple [ a b ]
  let tuple list a b
  ;; type tuple

  ;; if [ a b ] is NOT in the list of links
  if not ( member? ([list_of_links] of agent) tuple) [
    report tuple
  ]
end
```

Figura 23 - Função auxiliar que cria o tuplo

A função auxiliar '*create_link'* cria um tuplo com as duas KUs passadas como argumento, em que a que tiver menor valor fica em primeiro, para depois verificar que não há ligações duplicadas.

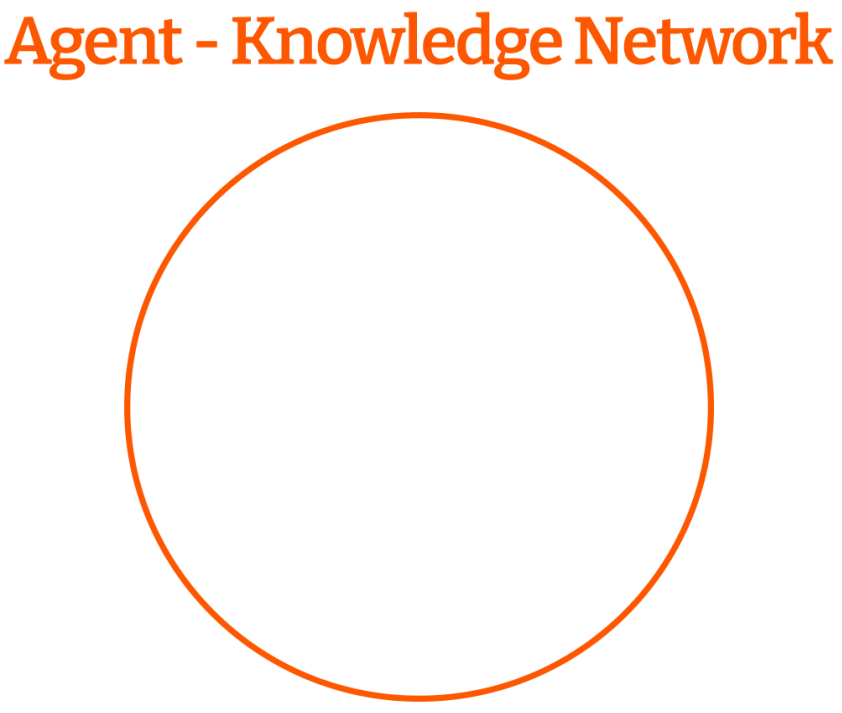


Figura 24 - Processo criação KNETs e Ligações entre KUs

7.4 Política de Atenção

A cada *tick* (unidade temporal em ABM, similar a um frame), os agentes 'focam' numa KU da sua rede. Há duas formas de trocar o foco, por *exploitation*, ou por *exploration*. A forma escolhida é aleatória, a chance é determinada pelos valores associados a cada género.

Os mecanismos *exploitation* / *exploration* funcionam da seguinte maneira:

- *Exploitation* : tendo em conta a KU atualmente em foco - guardada na variável *'focused_on'*, única para cada agente - é escolhida uma outra KU que esteja diretamente ligada com a original. Ou seja, a troca ocorre para uma KU compatível. Em código, isto foi alcançado ao iterar todas as ligações do agente (ordem aleatória) e, ao encontrar uma que contenha a KU a ser focada, o agente passa a focar o outro elemento da ligação.

```
;; switch focus to a directly connected KU
to exploit [agent]
  ;; check every tuple from list of links
  foreach [list_of_links] of agent [ tuple ->

    ;; if tuple has focused_ku the other element is going to be focused
    if member? [focused_ku] of agent tuple [

      ask agent[
        ;; we can switch tuples like this instead of having another condition
        let next_focused_ku abs(focused_ku - item 0 tuple - item 1 tuple)

        ;; compatibility
        let compare_compat compatibility decimal-to-binary focused_ku decimal-to-binary next_focused_ku

        ;; print
        type self type " is now focusing on " type next_focused_ku type ", "
        type compare_compat type "% compatible with " print focused_ku

        ;; switch focus
        set focused_ku next_focused_ku
        ;; exit loop
        stop
      ]
    ]
  ]
end
```

Figura 25 - Função para trocar o foco para uma KU diretamente conectada

❖ *Exploration* : tendo em conta a KU atualmente em foco, é escolhida uma outra KU que não esteja diretamente ligada com a original. A troca ocorre para uma KU não compatível. Na função '*explore'* é criada uma lista temporária com todas as KUs do agente, e a cada ligação que contenha a KU focada são removidos dessa lista os elementos constituintes (KU focada, KU compatível). No final o que temos é uma lista de todas as KUs que não estão ligadas à KU em foco (não são compatíveis).

```
;; switch focus to a directly UNconnected ku
to explore [agent]
  ;; for each agent
  ask agent[
    ;; list of KUs not connected to the KU being focused on currently
    let not_connected_kus kus

    foreach list_of_links [ tuple ->
      ;; check if focused ku is part of tuple
      if member? focused_ku tuple [
        ;; remove connected kus
        set not_connected_kus remove item 0 tuple not_connected_kus
        set not_connected_kus remove item 1 tuple not_connected_kus
      ]
    ]

    ;; no KU to switch to
    if length not_connected_kus = 0 [
      type "This KU is connected to all KUs, can't switch"
      stop
    ]

    ;; sort lists for cleaner output
    let k_kus sort filter [ ku -> not member? ku not_connected_kus ] kus
    set not_connected_kus sort not_connected_kus

    type "KUs connected to " type focused_ku type ": " print k_kus
    type "KUs NOT connected to " type focused_ku type ": " print not_connected_kus

    ;; next_focused_ku is a random ku that doesn't directly connect with focused_ku
    let next_focused_ku one-of not_connected_kus

    ;; print
    let compare_compat compatibility decimal-to-binary focused_ku decimal-to-binary next_focused_ku
    type self type " is now focusing on " type next_focused_ku type ", "
    type compare_compat type "% compatible with " print focused_ku

    ;; switch focus
    set focused_ku next_focused_ku
  ]
end
```

Figura 26 - Função para trocar o foco para uma KU não diretamente conectada

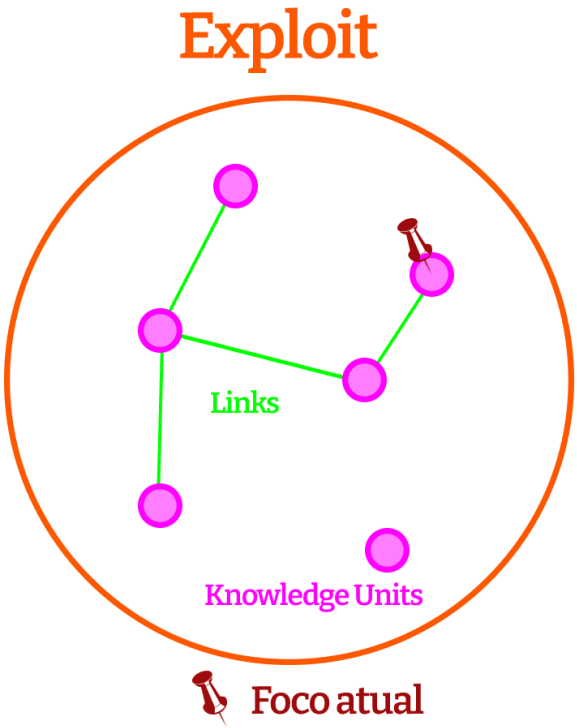


Figura 27 - Funcionamento Política de Atenção *'Exploit'*

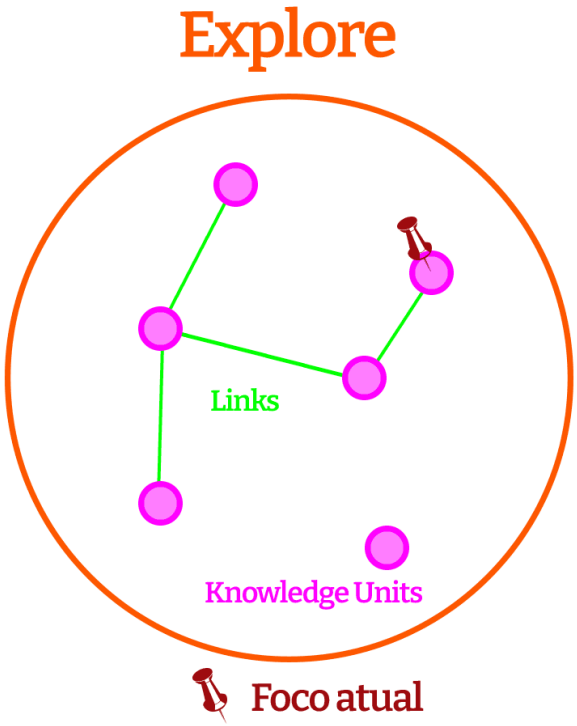


Figura 28 - Funcionamento Política de Atenção *'Exploit'*

7.5 Board

A *board* é inicializada com uma unidade de conhecimento aleatória e contém variáveis para guardar os dados (atuais e o histórico) das unidades de conhecimento, dos agentes que interagiram em cada tick, e das compatibilidades calculadas.

```
boards-own [
  curr_board     ; current kus on board
  board_history  ; history of kus posted on the board per tick

  agents_history ; all agents that sent kus to board
  curr_agents    ; agents that sent ku to board this tick

  ratio ; compat / incompat ratio, cant delete because of plot
  all_compatibilities
  compatibilities

  male_participation
  female_participation
]
```

Figura 28 - Variáveis da *board*

```
to setup-board
  create-boards 1
  [
    set shape "star"
    set color yellow
    set size 5
    set label "board"

    set agents_history [] ; save agent participation
    set curr_agents []

    set board_history lput (insert-item 0 [] first_ku) [] ; add first ku to board
    set curr_board []

    set all_compatibilities []
    set compatibilities []

    set ratio 0.5

    create-links-with other agents [
      set thickness 0.4
      if [gender] of [end1] of self = "m" [ set color yellow ]
      if [gender] of [end1] of self = "f" [ set color orange ]
    ]

  ]
end
```

Figura 79- Função que cria a *Board*

Foram desenvolvidas duas funções para calcular se uma mensagem é adicionada à board ou não. No método "*Compatibility*", a mensagem é adicionada se for compatível com qualquer outra do burst anterior, enquanto que no "*Attention Norm*" é obtida a mediana das compatibilidades entre todas as KUs do *burst* anterior e a atual em foco pela parte do agente, e a partir desse valor decide-se se o agente está ou não compatível, mediante a tendência geral de *exploit* e *explore* é calculada uma chance para decidir se o agente vai postar ou não.

```
to add-to-board-attention-norm-method [agent]
  ; let posts false
  let compatible false
  let compat 0
  ; type "agent " print [who] of agent
  ask boards [
    let compat_list []

    foreach last board_history [ ku_on_board ->
      set compat_list lput ( get-compat-as-decimal [focused_ku] of agent ku_on_board ) compat_list
    ]

    set compat median compat_list ; median dá sort automaticamente
    ; print compat
    if compat > c_threshold [ set compatible true ]

    let chance ( 1 - posting_chance )

    if ((compatible)     and (attention_norm = "exploit")) [ set chance posting_chance ]
    if ((not compatible) and (attention_norm = "explore")) [ set chance posting_chance ]

    ;; type attention_norm type " ; compat - " type compat type " : " type chance print "%"
    let r random-float 1
    if r < chance [
      if not member? [focused_ku] of agent curr_board [
        set curr_board lput [focused_ku] of agent curr_board
        set curr_agents lput [who] of agent curr_agents
        set compatibilities lput compat compatibilities
        set divergencies_from_first_ku lput (1 - get-compat-as-decimal [focused_ku] of agent first_ku) divergencies_from_first_ku
      ]
    ]

    set curr_board remove-duplicates curr_board
    ;;type "KUs on Board:" print curr_board
  ]
end
```

Figura 308 - Função método *Attention Norm*

7.6 Mecânica *Posting*

Mesmo que um agente esteja focado numa KU compatível com alguma na *board*, ela nem sempre é enviada. Para definir se ela é enviada ou não, temos primeiro que definir uma chance para tal acontecer com base na Norma de Atenção em vigor e com o estado de compatibilidade que o agente se encontra perante o burst anterior.

A Norma de Atenção é definida através do valor mais elevado da tendência das políticas de atenção *Exploit*/*Explore*, calculado com a seguinte fórmula:

$$Tendência\ Exploit\ = \boldsymbol{\Sigma\ Agentes\ em\ modo\ Exploit\ /\ \Sigma\ Agentes}$$

Para considerar um agente compatível (ou não), é calculada a mediana de todas as compatibilidades entre a KU em foco e as KUs do *burst* anterior. Se este valor for acima do limiar da compatibilidade, é considerado que o agente está em modo "Compatível".

	Exploit	*Explore*
Compatível	80%	20%
Não Compatível	20%	80%

Com esta mecânica de *posting*, abrimos a possibilidade dos agentes participarem na conversa mesmo se a sua KU em foco não for compatível com nenhuma na *board*.

Exemplo demonstrativo A:

1. Agente A está com o foco na KU ‘0101’;
2. A mediana das compatibilidades entre esta KU e as KUs do *burst* anterior é 45%;
3. 43% não ultrapassa o limiar de 50% para ser considerado compatível;
4. A Norma de Atenção dos agentes é ‘*Exploit’*;
5. A chance do agente fazer post da sua KU na board é de 20%

	Exploit	*Explore*
Compatível	80%	20%
Não Compatível	20%	80%

Exemplo demonstrativo B:

1. Agente B está com o foco na KU ‘0101’;
2. A mediana das compatibilidades entre esta KU e as KUs do *burst* anterior é 53%;
3. 53% ultrapassa o limiar de 50% para ser considerado compatível;
4. A Norma de Atenção dos agentes é ‘*Exploit’*;
5. A chance do agente fazer post da sua KU na *board* é de 80%

	Exploit	*Explore*
Compatível	80%	20%
Não Compatível	20%	80%

7.7 Métricas de Análise da Simulação

Norma de Atenção

A Norma de Atenção é a tendência prevalente entre agentes com a política de atenção exploit e explore. Ao acompanhar esta Norma conseguimos entender melhor o que se passa nas outras métricas, visto que ela afeta o comportamento de Posting dos agentes. Alterar as probabilidades da escolha da política de atenção por parte dos agentes altera esta métrica.

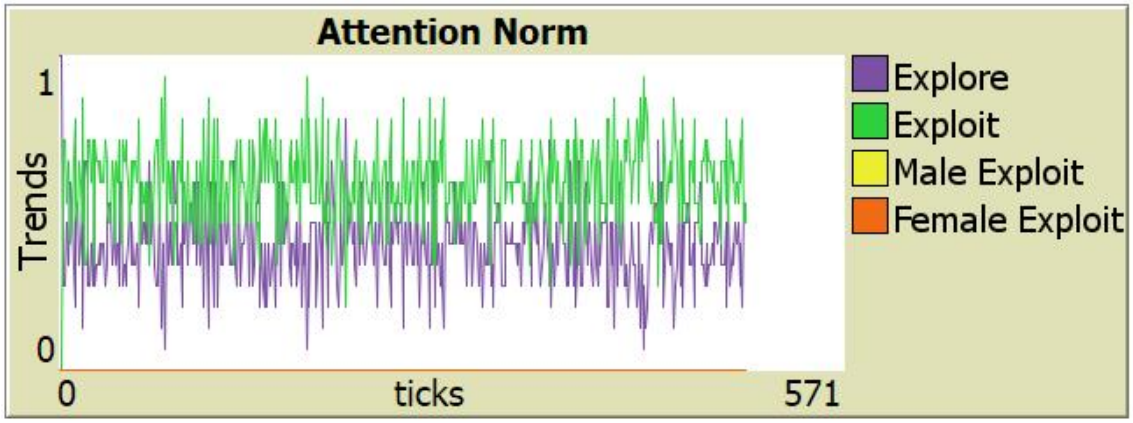


Figura 31 – Exemplo da Norma de Atenção visualizada ao longo da simulação

Divergência de Tópico

Com esta métrica analisamos o tamanho das sequências de mensagens em que os agentes ficaram a enviar mensagens *in-topic*, quando compatíveis com a mensagem inicial, ou *out-of-topic*, quando não são compatíveis com a mensagem inicial.

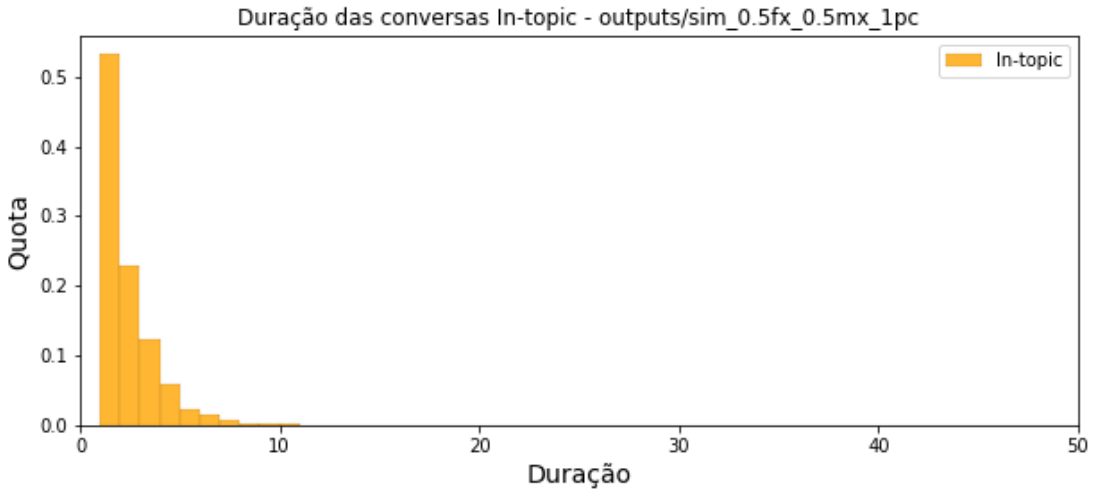


Figura 32 – Exemplo da distribuição da duração da conversa *in-topic*

Posts* por *burst

Em cada *burst* (1 por *tick*) podem ser enviados entre 0 e **Número de agentes** posts. Quantas mensagens são enviadas depende de variáveis como a norma de atenção corrente e das mensagens que foram trocadas no *burst* anterior. Medir quantas mensagens são enviadas por *burst* permite comparar a progressão e densidade da conversa entre os diferentes cenários.

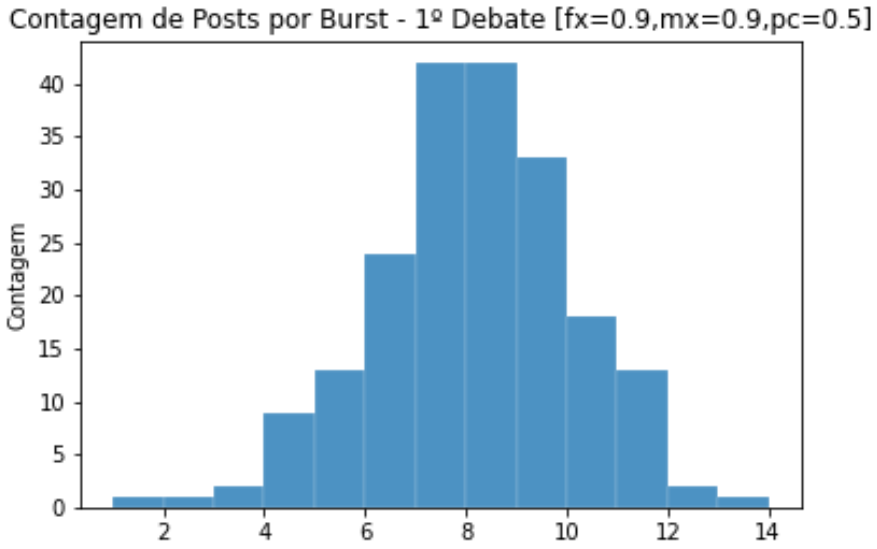


Figura 33 – Exemplo Contagem de *Posts* por *Burst*

Participação M/F

Analisar a participação dos géneros nos *bursts*. Com os dados desta métrica podemos avaliar os géneros dominantes, se houver.

7.8 Conector Python

A conexão com o NetLogo é feita através da biblioteca PyNetLogo, é necessário fornecer o endereço de instalação do NetLogo (pode ser facilitado programaticamente adicionando-o às variáveis de ambiente) e especificar a versão do NetLogo (6.2). Para carregar o modelo, é fornecido o endereço do mesmo e podemos estabelecer as variáveis que normalmente se alterariam na GUI:

```
# check if NetLogo env var exists
if "NetLogo" in os.environ:
    print(f"{os.environ['NetLogo']=}")
else:
    print("NetLogo not found in environment variables, adding it ... ")

    # substitute with your NetLogo installation path
    if platform == "linux":
        os.environ["NetLogo"] = "/home/paulo/Desktop/TFC/NetLogo-6.2.1-64/NetLogo 6.2.1"
```

✓ 0.5s

```
os.environ['NetLogo']='/home/paulo/Desktop/TFC/NetLogo-6.2.1-64/NetLogo 6.2.1'
```

Figura 34 - Adicionar caminho de instalação às variáveis de ambiente

```
# connect to netlogo
netlogo = pyNetLogo.NetLogoLink(
    gui=False,
    netlogo_home = os.environ['NetLogo'],
    netlogo_version="6.2",
)

def load_model():
    if platform == "linux":
        model_path = os.path.abspath(os.getcwd()) + "/files/knowledge_units_with_board.nlogo"
    elif platform == "win32":
        model_path = os.path.abspath(os.getcwd()) + "\\files\\knowledge_units_with_board.nlogo"

    netlogo.load_model(model_path)
    netlogo.command("set number_of_agents 5")
    netlogo.command("setup")

load_model()
```

Figura 35 - Ligar NetLogo e carregar modelo

Estando esta conexão feita, é possível realizar comandos como se fosse na própria consola da aplicação. Na seguinte figura, o comando *repeat_report* está a monitorizar a cada *tick* a participação por género e a guardar os dados num *DataFrame*. Estes dados estão prontos para serem representados graficamente.

```
netlogo.command("set female_prob_exploit 0.5")
netlogo.command("set male_prob_exploit 0.3")

iterations = 50
report = netlogo.repeat_report(\
    ["[male_participation] of one-of boards", "[female_participation] of one-of boards"],
    iterations)

fig, ax = plt.subplots()
ax.set_xlabel("Tick")
ax.set_ylabel("Gender Participation")
plt.title("Gender Participation / Tick")
sns.lineplot(data=report.iloc[:])
```

Figura 36 - Comandos exemplo NetLogo

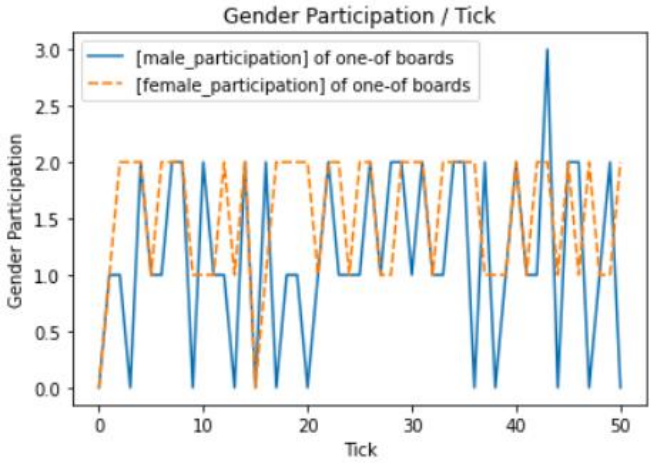


Figura 37 - Gráfico da participação por género a cada *tick*

No seguinte exemplo está representado o tamanho dos *bursts* e a sua *rolling average*.

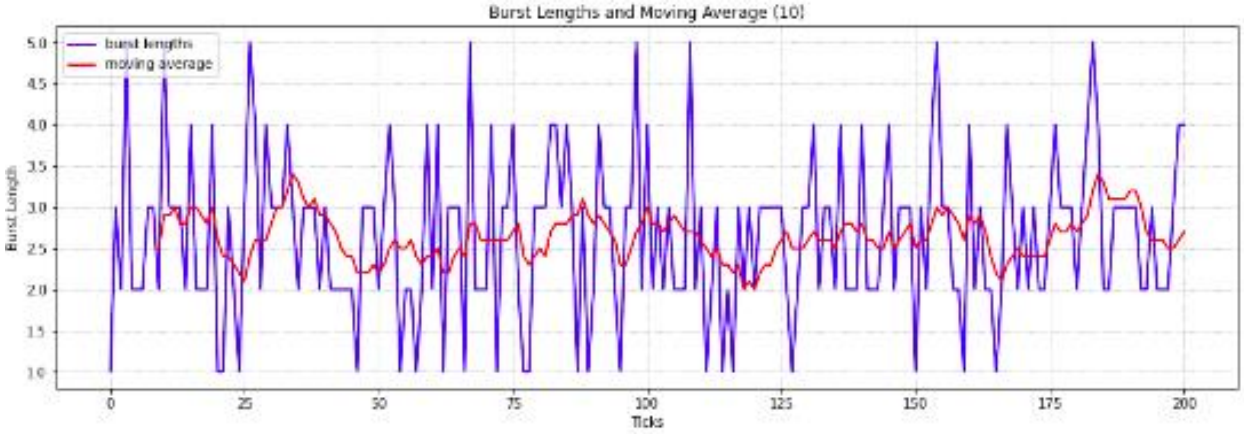


Figura 98 - Gráfico do tamanho dos *bursts*

A execução dos diferentes cenários é facilitada pelo Python, onde é possível descrevê-los numa estrutura de dados (neste caso um dicionário) e percorrê-los num *loop*.

No seguinte excerto de código é realizada toda a operação:

1. Itera os vários cenários
2. São selecionados os parâmetros para a simulação mediante o cenário
3. É criado o *folder* para guardar os dados (se não existir)
4. É executada a simulação durante o número de iterações pretendidas
5. São guardados os dados no ficheiro e em memória
6. São gerados os gráficos (histogramas com valores totais e normalizados)

```
for code, [f, m, fx, mx, pc] in new_scenarios.items():
    print(f"\n\nStarting Sim {code}\n\n")
    # sim options
    simulation_times = 50
    iterations = 300

    reported_columns = [
        "compat_ratio",
        "[ratio] of one-of boards",
        "length last [board_history] of one-of boards"
    ]

    netlogo.command(f"set female_prob_exploit {fx}")
    netlogo.command(f"set male_prob_exploit {mx}")
    netlogo.command(f"set posting_chance {pc}")
    netlogo.command("set number_of_agents 15")
    netlogo.command(f'set Method "Attention Norm"')

    # create path if it doesnt exist yet
    folder_path = f"outputs/sim_{fx}fx_{mx}mx_{pc}pc"
    if not os.path.exists(folder_path): os.makedirs(folder_path)
```

Figura 39 - Simular Cenários

```
    in_topic = []
    out_topic = []

    for i in range(1, simulation_times + 1):
        print(f"iter {i}")

        netlogo.command("setup")
        curr_path = f"{folder_path}/{i}.csv"

        report = netlogo.repeat_report(reported_columns, iterations)
        report = fix_col_names(report)

        # write to file
        report.to_csv(curr_path)

        in_, out_ = split_topics(report["Topic Divergence"].astype(int))
        in_topic += in_
        out_topic += out_

    plot_one_topic_histogram(in_topic, "in", folder_path, normalised = False)
    plot_one_topic_histogram(out_topic, "out", folder_path, normalised = False)
    plot_one_topic_histogram(in_topic, "in", folder_path, normalised = True)
```

Figura 40 - Simular cenários 2

8 Executar a Simulação

Para correr a simulação em si, só é necessário fazer *pull* do repositório e abrir o ficheiro knowledge_units_with_board.nlogo no NetLogo (versão 6.2).

Após instalar o Python (versão >= 3.8.8), há várias opções para correr o notebook:

- Abrir o ficheiro num IDE como o PyCharm ou o VSCode, que atualmente já vêm preparados para trabalhar com ficheiros deste tipo.
- Instalar o Jupyter Notebooks para executar o ficheiro localmente no *browser*.

Este *script* de Python usa (naturalmente) bibliotecas externas, para auxiliar a instalação das mesmas foi criado um ficheiro "requirements.txt" que permite instalar todas as dependências com um só comando "pip install -r requirements.txt". É possível criar um ambiente virtual de Python para não ter de se instalar todas as dependências no ambiente de Python base, as instruções para este passo estão descritas no início do *notebook*.

Execução

Uma execução exemplo da simulação utilizando a interface do NetLogo segue estes passos:

- Ajuste das variáveis da simulação (Interface):

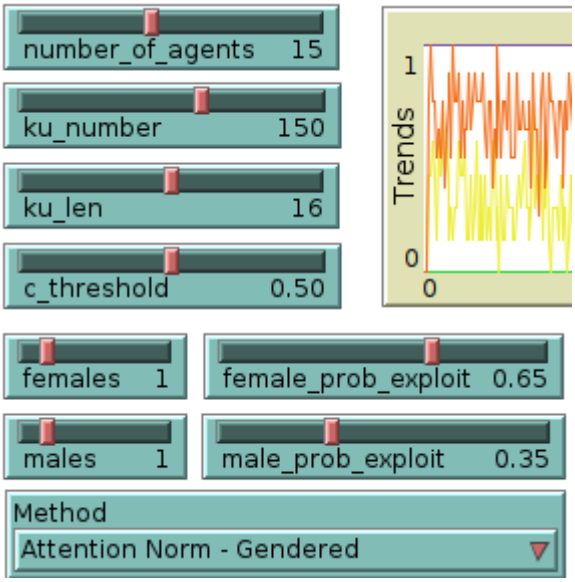


Figura 41 - Parâmetros

- Dar *Setup* à simulação, que envolve criar os agentes, a sua KNET e a *board* com a KU inicial fixa (255):

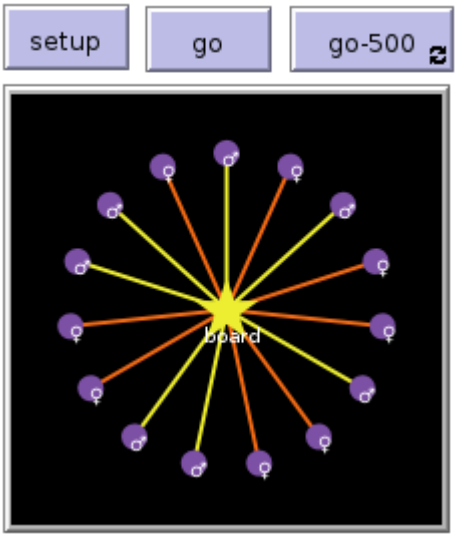


Figura 42 - Botões de *Setup* e *Go* e a representação gráfica dos agentes e da *board*

- Pressionar no botão '*go-500'* para simular 500 *ticks*. Os gráficos da interface irão começar a ganhar forma. Neles podemos observar:
    1. A compatibilidade média das mensagens enviadas por *burst*
    2. As divergências entre as mensagens enviadas e a primeira mensagem da conversa
    3. A divergência de tópico - se a maioria das mensagens forem compatíveis com a mensagem inicial ou não (divergem)
    4. Tendências *Explore* / *Exploit*
    5. Quantas mensagens foram enviadas por *burst*

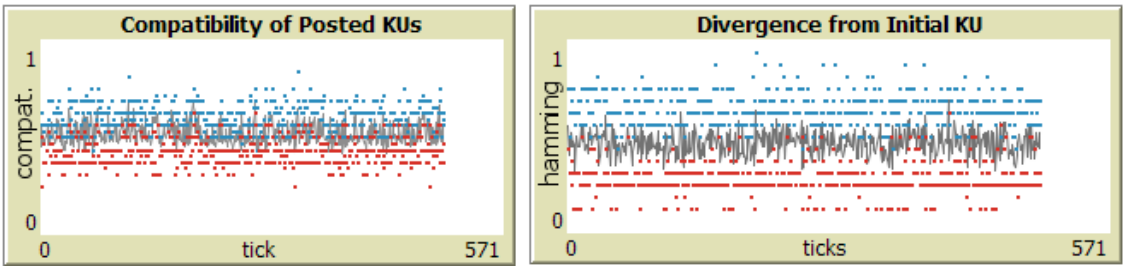


Figura 43 - Compatibilidades, Divergências da Primeira KU

Para executar o Jupyter Notebook que cria a conexão, carrega o modelo e gera o gráfico são necessários os seguintes passos:

- Instalar o Python (foi utilizada a versão 3.10.4)
- Instalar o NetLogo (versão 6.2)
- Colocar o endereço da instalação do NetLogo numa variável de ambiente chamada NetLogo
- Carregar o ficheiro do tipo notebook no IDE à escolha, juntamente com a pasta '*files'* com o ficheiro de simulação com a extensão .nlogo
- Executar pela ordem definida as células desejadas

9 Calendário

25 Nov 21: Definição do TFC, objetivos e métodos

28 Nov 21: Relatório Intercalar Sem 1

15 Dez 21: Primeira versão do modelo NetLogo que representa agentes ligados a um *chatroom*

20 Jan 22: Modelo inclui redes de conhecimento para cada agente, e dinâmica interna de atenção, e algumas das outras variáveis do modelo.

28 Jan 22: Relatório Intermédio

10 Fev 22: Modelo inclui as restantes variáveis consideradas como características de diversos grupos sociais. Nesta altura será possível instanciar um modelo com N agentes dum grupo social específico determinado pelos valores das variáveis consideradas.

1 Mar 22: Agentes já decidem e implementam as suas ações dependendo do seu estado interno e do que se está a passar na conversa

15 Mar 22: Ligação NetLogo e Python

15 Abr 22: Geração de dados de simulação (intra-grupo)

20 Abr 22: Design de Analíticas

24 Abr 22: Relatório Intercalar Sem 2

1 Mai 22: Implementação de Analíticas, indicadores estatísticos

15 Mai 22: Geração de dados de simulação inter-grupo

1 Jun 22: Validação do modelo com dados existentes (30% testing data)

15 Jun 22: Ajustes e revalidação concluídos

25 Jun 22 Volume de tese concluído

9 Set 22: Entrega Final (Época Especial)

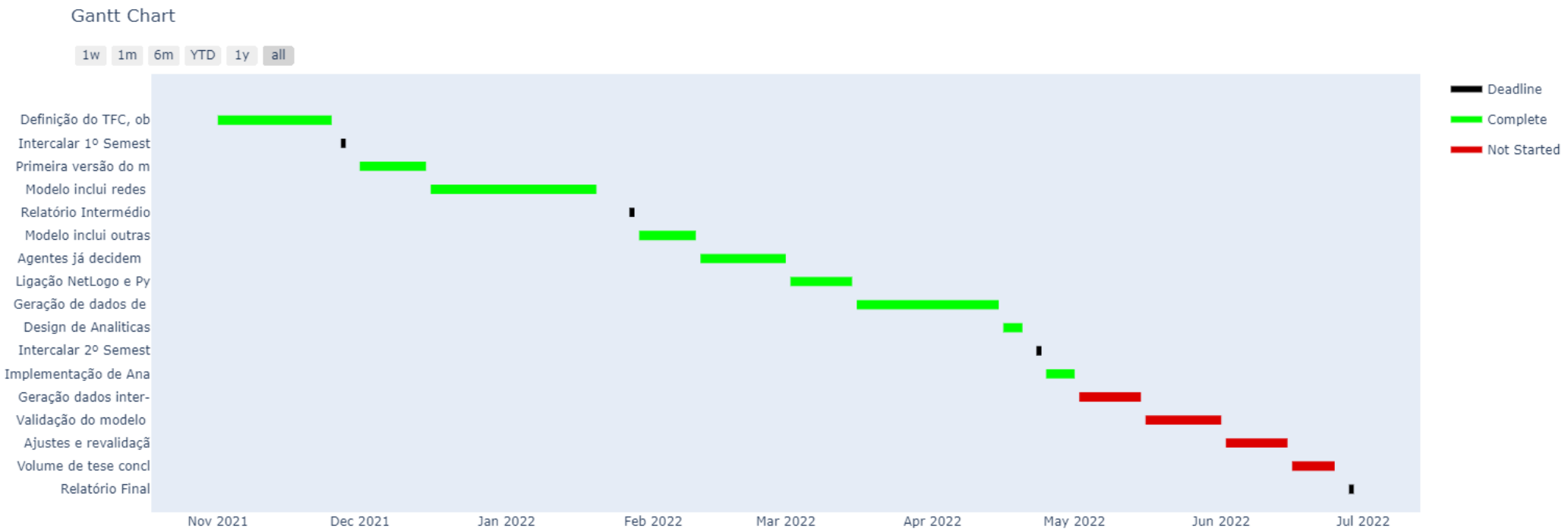


Figura 44 - Calendário

Conclusão

Nesta versão do modelo foi considerado um número muito reduzido de variáveis, no entanto consegue-se evidenciar fenómenos coletivos interessantes relativamente à participação na conversa e à duração com que perdura uma conversa dentro do mesmo tópico.

Naturalmente, diferentes configurações geram diferentes resultados, foi evidenciada uma relação entre o aumento da polarização dos comportamentos de troca de foco (exploit vs explore) e o aumento do número de agentes a participar concorrentemente. A taxa de escolha da troca de foco também influencia quanto perdura uma conversa dentro/fora do mesmo tópico (in-topic/out-of-topic).

Devido a uma falta de tempo não foi possível progredir mais no desenvolvimento deste modelo no âmbito do TFC, mas ele está preparado para continuar a evoluir. As últimas implementações - métricas de análise - serão extremamente úteis para os próximos passos, onde é necessário ter ferramentas para avaliar se o modelo está a produzir resultados similares aos observados nos dados existentes.

Bibliografia

[DEISI21] DEISI, Regulamento de Trabalho Final de Curso, Set. 2021.

[TaWe20] Tanenbaum,A. e Wetherall,D., *Computer Networks*, 6ª Edição, Prentice Hall, 2020.

[Zh20] Zhang, Y., Wang, Y., Chen, T., & Shi, J. (2020). Agent-based modeling approach for group polarization behavior considering conformity and network relationship strength. *Concurrency and Computation: Practice and Experience*, *32*(14), e5707.

[Bo02] Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the national academy of sciences*, *99*(suppl 3), 7280-7287.

[La10] Laskowski, K. (2010, July). Modeling norms of turn-taking in multi-party conversation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (pp. 999-1008).

[JoYo17] Srour, F. J., & Yorke-Smith, N. (2017, May). An initial study of agent interconnectedness and in-group behaviour. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation* (pp. 105-120). Springer, Cham.

[Sh21] Schlaile, M. P., Zeman, J., & Mueller, M. (2021). It's a match! Simulating compatibility-based learning in a network of networks. In Memetics and Evolutionary Economics (pp. 99-140). Springer, Cham.

[AbLi21] Abbott, R., & Lim, J. S. (2021). PyLogo: A Python Reimplementation of (Much of) NetLogo.

[JaKw18] Jaxa-Rozen, M., & Kwakkel, J. H. (2018). Pynetlogo: Linking netlogo with python. Jasss, 21(2).

[Fu15] Fuhse, J. A. (2015). Theorizing social networks: The relational sociology of and around Harrison White. International Review of Sociology, 25(1), 15-44.

[OkSm08] Okamoto, S., & Smith, J. S. S. (2008). Constructing linguistic femininity in contemporary Japan: scholarly and popular representations. Gender & Language, 2(1).

[BaLa08] Barzilay, R., & Lapata, M. (2008). Modeling local coherence: An entity-based approach. Computational Linguistics, 34(1), 1-34.

[ULHT21] Universidade Lusófona de Humanidades e Tecnologia, www.ulusofona.pt,

Glossário

LEI Licenciatura em Engenharia Informática

TFC Trabalho Final de Curso

ABM Agent Based Modelling (Modelação Baseada em Agentes)

MAS Multi-Agent System (Modelação Baseada em Múltiplos-Agentes)