Universidade de Aveiro Departamento de Física
2011

Bruno Filipe dos Santos Faria

Algoritmos imunológicos artificiais e sua implementação em *GPU*

Universidade de Aveiro Departamento de Física
2011

Bruno Filipe dos Santos Faria

Algoritmos imunológicos artificiais e sua implementação em *GPU*

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Física, realizada sob a orientação científica do Doutor Fernão Vístulo de Abreu, Professor Auxiliar do Departamento de Física da Universidade de Aveiro

Apoio financeiro da FCT no âmbito do projeto BI/FIS-21-2010/MOD-FRUST-CEL

o júri

presidente

Prof. Dr. Manuel Almeida Valente
Professor associado do Departamento de Física da Universidade de Aveiro

Prof. Dr. Fernão Rodrigues Vístulo de Abreu
Professor auxiliar do Departamento de Física da Universidade de Aveiro

Dra. Berta Maria Barbosa Neto
Investigadora, Nokia Siemens Networks

agradecimentos

Quero agradecer a todas as pessoas que contribuíram para este trabalho, em particular: ao Prof. Dr. Fernão Abreu pelo seu imenso apoio e paciência e à Patrícia Mostardinha. Agradeço ainda o apoio assim como a disponibilidade de diversas pessoas no Departamento de Física, que ajudaram a tornar este projeto possível. Quero agradecer também ao Prof. Dr. André Zúquete por todas as discussões bastante produtivas. Por fim agradeço ainda a disponibilidade do júri para arguir esta tese.

palavras-chave Sistemas imunológicos artificiais, seleção negativa, discriminação *self-nonself*, paralelização, GPU, frustração celular.

resumo Neste trabalho é apresentado um modelo para a deteção de elementos estranhos, recorrendo ao mecanismo de frustração celular. É discutida a paralelização do algoritmo em unidades de processamento gráfico. Mostra-se que desta forma é possível atingir ganhos computacionais de 406 vezes relativamente à versão sequencial equivalente. Neste trabalho são ainda apresentados resultados que demonstram que os algoritmos de frustração celular conseguem realizar discriminação "*self-nonself"* perfeita em espaços de dimensão arbitrária

keywords Artificial immune systems, negative selection, self-nonself discrimination, parallelization, GPU, cellular frustration.

abstract In this work a model for nonself detection is presented, based on the cellular frustration mechanism. A GPU implementation is discussed that achieved computational gains of 406 times faster execution times as compared to its sequential implementation. I also present results showing that perfect self-nonself discrimination can be achieved in these models for spaces in arbitrary dimensions.

Índice

Capítulo 1: Introdução

A deteção de comportamentos anómalos e intrusões continua a ser um enorme desafio com muitas potenciais aplicações. A dificuldade resulta não só do facto dos intrusos disfarçarem as suas intenções, mas também do número de indícios requeridos para os identificar poder ser muito variado. Além disso, um sistema de deteção de intrusões deve ser eficiente e apresentar pequenos custos, o que na prática é difícil de concretizar. Um exemplo evidente é o da segurança nos aeroportos. Neste caso, o número de indícios conhecidos de potencial perigo pode ser extremamente elevado. Para além disso, podem existir atividades perigosas que importaria sinalizar, mas cujos indícios podem ser difíceis de valorizar por falta de conhecimento. A tendência natural das estratégias de segurança é a de aumentar a complexidade dos sistemas de deteção e a sua precisão, o que pode acarretar custos mais elevados e até mesmo invasibilidade.

O problema da deteção de intrusos é muito geral. Problemáticas semelhantes podem ser encontradas na área de segurança informática, como por exemplo, na proteção dos sistemas informáticos de ataques de vírus. Neste caso, a crescente complexidade dos antivírus tem impacto não só no tempo computacional requerido, mas também na memória exigida, o que representa custos para o utilizador. Por outro lado, os antivírus actuais apenas são eficazes contra ataques de vírus conhecidos, o que significa que o utilizador se encontra completamente desprotegido contra novos ataques.

É a capacidade de atacar invasores no sistema imunológico que torna os algoritmos neles inspirados tão desejáveis. Estes algoritmos são designados por algoritmos imunológicos artificiais. Os algoritmos imunológicos artificiais mais conhecidos e comummente aceites, são os algoritmos de seleção negativa (ASN). Estes algoritmos foram desenvolvidos por Stephanie Forrest e pelos seus colaboradores[1]. Estes algoritmos procuram detetar padrões estranhos contidos num conjunto de dados. A identificação do padrão estranho é efetuada com recurso a um conjunto de detetores. Os detetores são responsáveis por identificar o maior número de padrões estranhos possível. No entanto, é neste conjunto de detetores que reside a maior desvantagem dos algoritmos de seleção negativa. Com efeito, boas taxas de deteção (> 90%) requerem um número elevado de detetores. Na prática isto implica que o uso de um conjunto de detetores limita as taxas de deteção a ~90%[2, 3].

Um outro algoritmo que tem os mesmos objetivos que os algoritmos de seleção negativa é o algoritmo de frustração celular. Este algoritmo, proposto pelo orientador desta tese, baseia-se no princípio de que as interações entre as células do sistema efetuam tempos de ligação pequenos. Por sua vez, a interação com um qualquer agente estranho originará tempos longos, o que permite ao sistema detetar o agente estranho com sucesso. Um aspeto importante neste algoritmo é que o número de detetores requeridos não é proporcional ao número de padrões estranhos.

O trabalho aqui apresentado vem dar continuidade ao trabalho desenvolvido pelo orientador desta tese, e de um antigo aluno de mestrado. Nesse trabalho, mostrou-se que era possível comprimir a informação contida nas listas de interação e manter boas taxas de deteção. Este trabalho tenta ir mais além, mostrando não só que é possível comprimir a informação de uma qualquer lista, mas também que é possível efetuar uma deteção perfeita.

Nos capítulos a seguir, irei detalhar os problemas que surgiram no decorrer deste trabalho, assim como, as soluções encontradas para os ultrapassar. No entanto, em primeiro lugar é apresentada uma breve introdução sobre a visão atual do sistema imunológico, e as motivações que levaram ao aparecimento dos algoritmos imunológicos artificiais (capítulo 2).

No terceiro capítulo, é descrito o mecanismo que levou à formalização do modelo de frustração celular, e são apresentadas as regras para comprimir a informação contida nas listas de interação.

No quarto capítulo é descrita a implementação do algoritmo de frustração celular em unidades de processamento gráfico (do inglês *GPU*). São apresentados os problemas inerentes a uma paralelização simples do algoritmo, e descrita uma estratégia para os ultrapassar. Para além disso, é ainda analisado o ganho em tempo computacional desta implementação.

No capítulo 5, são apresentados e discutidos os vários resultados provenientes da deteção de agentes estranhos, considerando para o efeito as várias métricas descritas no capítulo 3. Por último, apresentam-se as conclusões relativas a todo o trabalho, e discutem-se possibilidades de melhoria futura.

Capítulo 2: Motivação Imunológica

A modelação do sistema imunológico é uma área de grande interesse dada a necessidade em alcançar uma melhor compreensão sobre como os vários fenómenos imunológicos (a memoria, a aprendizagem e o reconhecimento de antigénios) emergem das interações celulares. De modo a modelar o sistema imunológico é necessário conhecer os seus constituintes e as regras de interação que os governam. A forma como o sistema imunológico protege o corpo humano da doença é muito complexa. A complexidade resulta dos vários processos e estruturas no organismo que permitem que este combata agentes patogénicos muito diversos tais como bactérias, parasitas, vírus e fungos. Para poder combater estes agentes, o sistema imunitário necessita de os identificar e de os distinguir das células saudáveis que compõem o organismo. O sistema imunitário enfrenta assim dois problemas:

- A identificação dos agentes patogénicos
- A eliminação pronta dos agentes patogénicos sem no entanto causar danos ao organismo.

Neste trabalho estudar-se-á um mecanismo que permite ao sistema imunológico realizar a tarefa de discriminação entre elementos que pertencem ao corpo ("*self")* e os elementos que a ele não pertencem ("*nonself"*) [4, 5]. O sistema imunitário pode ser dividido em duas classes: o sistema imunitário não específico (inato), e o sistema imunitário específico (adaptativo). O sistema imunitário não específico tem um mecanismo de defesa que não é particular a um determinado agente patogénico, muito embora possa ter evoluído por seleção natural nesse sentido. Este sistema providência a primeira linha de defesa no combate a um agente patogénico e é constituído por barreiras anatómicas (pele, sobrancelhas, etc), secreções (saliva, lágrimas, etc) e células fagocíticas, tais como macrófagos, neutrófilos e células exterminadoras naturais [6]. Os mecanismos de defesa permitem ao sistema imunitário não específico, eliminar a maior parte dos microrganismos. No entanto, quando um microrganismo ilude o sistema imunitário não especifico, ou não é eliminado por este, pode ainda ser acionada uma resposta do sistema imunitário específico. Isto porque, as células do sistema inato interagem com as células do sistema adaptativo, estimulando-as a promover uma resposta. Além disso, as células do sistema adaptativo possuem a capacidade de reconhecer padrões muito mais variados e desenvolver respostas muito mais específicas.

O sistema imunitário adaptativo é maioritariamente constituído por linfócitos tais como células B e T, moléculas de anticorpos, e outras moléculas produzidas pelos linfócitos. Como o seu nome sugere a resposta deste sistema pode ganhar especificidade em relação a um determinado patogénio, ou seja adaptar-se de forma a combatê-lo mais eficazmente. Para reconhecer antigénios o sistema imunitário conta com as células B e T, as quais possuem recetores na sua superfície. No caso das células B estes são denominados de anticorpos (BCR), e no caso das células T denominam-se simplesmente de recetores T (TCR). Ambos os recetores são responsáveis pelo reconhecimento do antigénio, ou pelo menos da parte reconhecível deste, que se denomina de epítopo.

O reconhecimento do antigénio, por parte das células B e T é efetuado com base na afinidade entre o recetor e o antigénio, e processa-se de acordo com o tipo de célula. No caso das células B, o reconhecimento do antigénio é efetuado quando este se encontra na sua forma livre (solúvel), no sangue ou linfa. Em contraste, as células T reconhecem o antigénio na sua

forma processada, ou seja, quando é apresentado como um fragmento de peptídeo pelas células apresentadoras de antigénios (APC). A apresentação destes fragmentos é feita recorrendo a moléculas de MHC (complexo principal de histocompatibilidade), as quais recolhem fragmentos de peptídeos no ambiente intracelular da célula APC e os transportam para a sua superfície para apresentação aos recetores das células T. Estes fragmentos resultam do natural metabolismo da célula. No caso de células dendríticas e macrófagos, dado que estas ingerem materiais diversos (por exemplo, células ou restos de células) durante a sua função de recolha no corpo, estes peptídeos podem ter origem patogénica. Porém, na maior parte das situações podem provir do natural funcionamento do corpo. O sistema imunitário deve por isso ser capaz de distinguir os peptídeos de acordo com a sua origem.

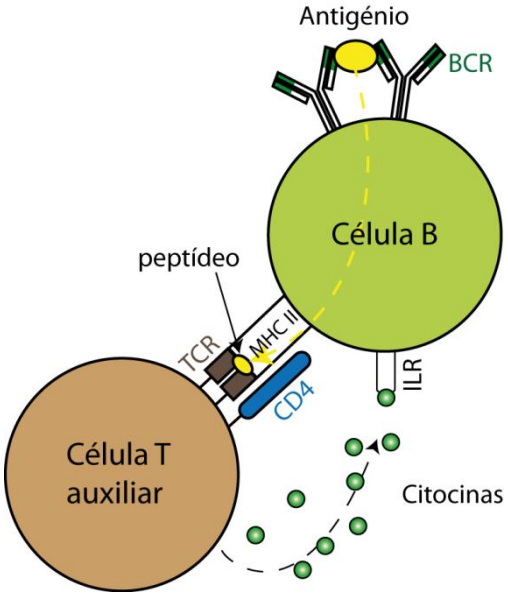


Fig. 2.1– Representação dos elementos de superfície de uma célula T e uma APC (célula B). De notar a ligação do recetor da célula B (BCR) ao antigénio e a respetiva apresentação do fragmento de peptídeo ao recetor da célula T auxiliar (TCR), recorrendo ao complexo principal de histocompatibilidade de classe II.

As moléculas MHC dividem-se em duas classes. As moléculas da classe I são especializadas em apresentar as proteínas sintetizadas nas células, enquanto as da classe II estão especializadas na apresentação de fragmentos de moléculas recolhidos do ambiente extracelular. As moléculas da classe II apenas se encontram em células de apresentação de antigénios (APC), tais como as células B, os macrófagos e as células dendríticas. Ambas as classes de moléculas MHC ligam-se a peptídeos sendo estes apresentados às células T nos nódulos linfáticos.

As moléculas de classe I apresentam os peptídeos às células T citotóxicas (TCL-linfócitos citotóxicos T). Uma vez ativadas estas células T citotóxicas destroem as células apresentando o complexo constituído pelo peptídeo e a molécula MHC classe I.

As moléculas classe II por sua vez apresentam os peptídeos às células T auxiliares (Th, do inglês “T helper”), as quais segregam citocinas se forem ativadas (i.e., se se conseguirem ligar ao complexo MHC-peptídeo para desencadear uma resposta imunitária). As citocinas são proteínas que regulam a intensidade e a duração da resposta imunitária, exercendo para tal um conjunto de efeitos nos linfócitos e outras células imunitárias. Por exemplo, o reconhecimento de um antigénio em conjunto com a ligação a uma célula Th, por parte de uma célula B, leva ao

aparecimento de um sinal de co-estimulação e à libertação de citocinas. As citocinas, irão iniciar a divisão e a diferenciação de ambas as células B e Th, em células efetoras e de memória. As células efetoras das células B denominam-se por plasma de células B, e tem a função de segregar grandes quantidades de anticorpos. Os anticorpos permitem ao sistema imune identificar as moléculas como estranhas. Um complexo antigénio-anticorpo, ou uma célula estranha ligada a anticorpos é rapidamente eliminada por células fagocíticas, tais como macrófagos.

São vários os fenómenos que caracterizam a resposta imunitária. No entanto, os mais notáveis são a exibição de memória e a discriminação *self-nonself (próprio não próprio)*. A exibição de memória manifesta-se, quando o sistema imunitário responde a uma segunda invasão de um mesmo agente patogénico. Verifica-se que, mesmo que a resposta à primeira invasão tenha demorado alguns dias, a segunda resposta é quase imediata. Diz-se neste caso que, o sistema adquiriu imunidade ao patogénio.

A discriminação *self-nonself* é provavelmente o desafio mais extraordinário que o sistema imunitário tem de enfrentar. O sistema imunitário tem de ser capaz de reconhecer o maior número de antigénios possível, evitando ao mesmo tempo provocar autoimunidade, ou seja evitando dirigir uma resposta imunitária contra as células do próprio corpo - o *self*. De forma a evitar a autoimunidade, o sistema imunitário desenvolveu um período de educação para os linfócitos T, que migram da medula óssea para o timo. O processo de educação no timo é um processo composto por duas etapas. Numa primeira etapa as células T são selecionadas positivamente de acordo com a sua capacidade de se ligarem a moléculas MHC [4]: as células que não se conseguirem ligar são mortas por apoptose (morte programada da célula).

As células T que forem selecionadas positivamente têm ainda de enfrentar uma segunda etapa, denominada de seleção negativa. Esta etapa permite que o sistema imunitário não reaja contra as células do próprio corpo, do *self*. De forma a ganhar esta tolerância, as células T são submetidas a um processo de maturação, durante o qual lhes são apresentados peptídeos tipicamente expressos por células do corpo. Esta apresentação é feita recorrendo à molécula MHC, presente nas células APC. Se as células T se ligarem fortemente aos peptídeos do *self*, ser-lhes-á fornecido um sinal de apoptose. O que por sua vez significa, que apenas as células T que se liguem a peptídeos *nonself* permanecerão. Note-se no entanto que os processos de educação no timo não garantem que os linfócitos sejam muito reativos contra peptídeos *nonself*. Na secção seguinte, mostrar-se-á que efetivamente pode não ser evidente que isso assim seja. Para tal, será apresentada uma classe de modelos computacionais que foi inspirado nos processos de seleção negativa, que ocorrem no timo, e que têm o mérito de ser alguns dos poucos modelos quantitativos que os descrevem e que estudam as suas consequências.

2.1 Algoritmos de seleção negativa (ASNs)

O sistema imunitário é um sistema natural que protege o corpo humano da doença. Este tipo de proteção inspirou o aparecimento dos sistemas imunológicos artificiais (SIA). De todos os mecanismos que são explorados dentro dos SIAs, os que tem ganho mais reconhecimento são, o de seleção negativa, o de seleção clonal, e o de rede imunológica [7].

O processo de seleção negativa foi um dos primeiros a ser considerado. Este processo descreve a visão, de como o sistema imunológico faz a seleção das células T. A seleção negativa artificial é uma imitação computacional do processo de seleção de células T. Este modelo foi introduzido pela primeira vez pela Forrest et al [1], como um meio de detetar sequências binárias de vírus em sistemas computacionais. É definida uma população de detetores para desempenhar o trabalho das células T. Estes detetores são sequências binárias de comprimento

fixo. É ainda definida uma regra para verificar a correspondência entre duas quaisquer sequências. A correspondência equivale à correspondência entre um linfócito e um antigénio.

Cada possível detetor gerado aleatoriamente é comparado com o conjunto do *self*. O conjunto de *self* é análogo às proteínas do *self* armazenadas no timo, na forma de conter elementos do *self* contra os quais são testados os detetores. Todos os detetores que igualarem uma determinada sequência do conjunto do *self*, são excluídos do conjunto de detetores. Desta forma, novas sequências podem ser recolhidas de um sistema a ser vigiado. Estas sequências são traduzidas para a forma binária, e comparadas com o conjunto de detetores. Se a sequência a ser testada igualar algum dos detetores, então é garantido que a sequência pertence ao conjunto de *nonself* e algum tipo de ação deve ser tomada.

Se fosse exigida uma correspondência exata entre a sequência do detetor e a sequência a ser testada, então iria ser necessário um detetor para cada sequência do *nonself*. O que imporia um elevado peso computacional, e tornaria o uso do algoritmo impraticável. Em vez disto, a deteção é probabilística. Apenas é necessária que exista correspondência num conjunto de bits *r*. O valor de *r* é conhecido como limite de correspondência. A figura 2.2 mostra a existência de correspondência quando *r* é menor que 5.

0 1 1 **1 1 0 1** 0
0 1 0 **1 1 0 1** 1

Fig. 2.2-Exemplo da correspondência entre duas sequências quando r ≤ 4. De notar que quando r > 4 não existe correspondência entre as sequências.

A deteção probabilística torna os algoritmos de seleção negativa vulneráveis. A vulnerabilidade resulta de erros na classificação de sequências, classificação em *self* quando deveria ser *nonself*. As sequências que dão origem aos erros de classificação são designadas por buracos [3]. Um exemplo dos buracos encontra-se ilustrado na figura 2.3. A região de deteção de cada detetor encontra-se ilustrada por círculos, o conjunto de *self* por uma cor alaranjada, e os buracos por uma região a vermelho.

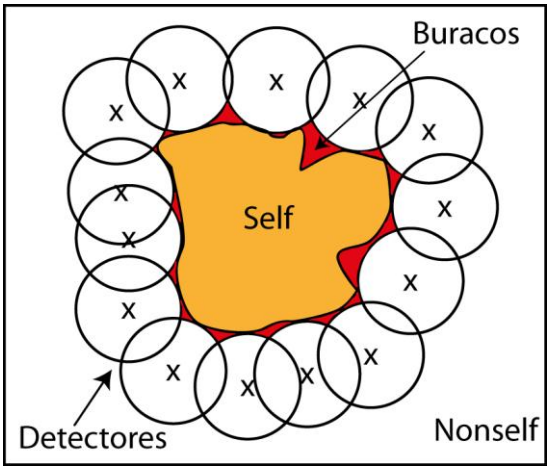


Fig. 2.3- Ilustração da região de *nonself* coberta por detetores em torno do *self*. De notar o aparecimento de regiões (a vermelho), onde não é possível colocar nenhum detetor. Estas regiões resultam da eliminação de detetores candidatos durante a etapa de censura por igualarem sequências do self.

O aparecimento de buracos deve-se à regra usada, e à similaridade entre sequências do *self* e do *nonself*. Estes buracos, resultam da eliminação de detetores candidatos por igualarem sequências do *self*. Por exemplo, considere-se o conjunto de *self* definido por $S = \{110, 010\}$, e que a regra usada é a *r-contiguous bits*[4, 5], com *r=2*. Nesta situação, é impossível gerar um detetor para uma sequência do *nonself*, definida como 011, pois o detetor irá igualar uma sequência do *self*. Com o aumento do tamanho da sequência para um mesmo valor de *r*, aumenta também o número de buracos. Isto é uma desvantagem dos algoritmos de seleção negativa. No capítulo seguinte, é descrito um algoritmo alternativo que procura combater esta desvantagem.

Capítulo 3: Algoritmos de frustração celular

Os algoritmos de frustração celular (AFCs) são uma alternativa aos algoritmos de seleção negativa (ASNs). Estes algoritmos, procuram atingir a mesma capacidade de deteção de anomalias e reconhecimento de padrões que, a observada no sistema imunitário. Neste sentido, pode-se dizer que ambos os algoritmos têm os mesmos objetivos, mas diferem no método para os atingir. Os ASNs acreditam que, a capacidade do sistema imunitário resulta da ação individual e independente de cada célula. Já no caso dos AFCs, esta capacidade emerge da ação correlacionada de todos os constituintes.

Uma das maiores dificuldades para definir algoritmos imunológicos resulta, da necessidade em conciliar uma alta reatividade contra elementos estranhos, aliada à manutenção de, baixa reatividade ou tolerância, em relação a elementos do próprio corpo. Nos ASN, procuram-se restringir os domínios de atuação de cada célula T, de forma, a não conterem elementos do próprio corpo. Como se viu no capítulo anterior, isso tem o custo de se criarem buracos no espaço dos padrões que podem ser apresentados. A ideia dos AFCs é diferente. Neste caso, assume-se à partida que a reatividade potencial das células do sistema imunitário é máxima, sendo garantida a tolerância em relação ao próprio corpo através de um mecanismo de frustração celular, que se discutirá de seguida.

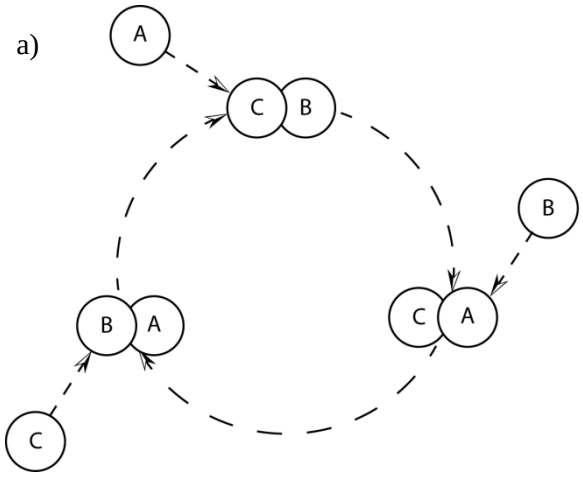
3.1 O conceito de frustração celular

A grande alteração conceptual introduzida nos AFCs reside na forma, de como as reações celulares são despoletadas. Nos ASN, cada célula reage instantaneamente sempre que reconheça um padrão no seu domínio de deteção. Ao contrário, nos algoritmos de frustração celular as reações celulares são efetuadas como se fossem decisões [8]. Estas decisões podem demorar tempo a serem desencadeadas e esse tempo será crucial para gerar um novo mecanismo de reconhecimento e tolerância. De forma a ganhar uma melhor compreensão deste fenómeno, considere-se o exemplo presente na figura 3.1. O exemplo mostra um sistema composto por 3 agentes, A, B e C. Neste sistema, os agentes interagem de acordo com uma lista de interação, procurando sempre, manter-se ligados a agentes que estejam no topo da sua lista de interação. Neste modelo, assume-se que os agentes preferem estar ligados entre si, do que separados, mas não podem manter ligações estáveis com mais do que um outro agente de cada vez. Os agentes são obrigados a tomar decisões sempre que, temporariamente contactarem com mais do que um agente.

Nesse caso, a decisão recai sobre a qual dos agentes manter-se ligado. Todos os agentes executam uma dinâmica semelhante, ainda que usando listas de interação diferentes. Assim, se o agente B estiver ligado ao agente A, e interagir com o agente C, então deverá quebrar a sua ligação anterior, e formar uma nova com o agente C. O mesmo raciocínio pode ser aplicado ao novo par conjugado e aos seguintes. Desta forma, pode-se verificar que no sistema descrito na figura 3.1-a, os agentes trocarão continuamente de par não conseguindo estabelecer pares estáveis. Neste caso os pares que se formam serão sempre de curta duração, pois existe sempre um agente não ligado que pode destabilizar o par, frustrando a dinâmica.

A dinâmica frustrada surge assim como um resultado natural deste tipo de sistemas, no qual os tempos de vida dos pares conjugados, são uma propriedade emergente da dinâmica. Estes tempos de vida característicos dependem intrinsecamente da configuração da população, dos agentes presentes e do seu comportamento.

a)



b)

A	B	C
B	C	A
C	A	B

Fig. 3.1- Esquema de uma dinâmica frustrada (a).Neste exemplo não se forma nenhum par estável. De acordo com a lista de interação definida na tabela à direita (b), o agente A prefere estar ligado ao agente B, do que com ao C. Como resultado se estiver ligado a C e lhe for dada a oportunidade de interagir com B, ele terminará a ligação anterior e ligar-se-á ao agente B. Seguindo este raciocínio pode-se concluir que um sistema deste género, apenas poderá formar pares transientes.

Este tipo de dinâmica é caracterizado por um conjunto de propriedades e efeitos bastante interessantes. Um destes exemplos é o facto de, a diminuição da reatividade individual poder levar a um aumento da reatividade do sistema como um todo. Este efeito pode ser observado quando se elimina a possibilidade de um agente reagir com outro. Por exemplo se se eliminar o agente B da lista do agente C, e dado que o agente C fica impossibilitado de interagir com o agente B, possibilita-se a formação de um par AB estável. Se admitirmos que as reações requerem um certo tempo para ser despoletadas, então a reatividade aumentou porque o par AB, anteriormente instável, deixou de poder ser destabilizado e por conseguinte pôde desencadear uma reação. A situação pode parecer paradoxal, pois reduziu-se a reatividade de um agente, mas aumentou-se a reatividade do sistema como um todo.

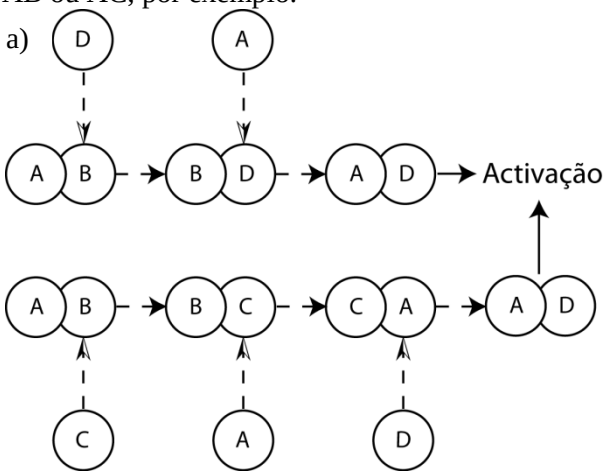
Uma outra propriedade interessante surge quando se introduz um elemento estranho no sistema. Nesta situação, podem existir sistemas para os quais se aumenta a reatividade do sistema em direção ao agente estranho. Para uma melhor compreensão deste efeito, considere-se que é introduzido um agente D no sistema ABC descrito anteriormente. Considere-se ainda, que este agente é semelhante ao agente C, no sentido em que tem a mesma lista de interações. No entanto este agente D é diferente do agente C no sentido em que apresenta um ligando diferente. Esta diferença permite aos restantes agentes efetuar uma decisão que distinga os agentes C e o D (figura 3.2).

Neste sistema, o agente C continua frustrado tal como no sistema ABC. No entanto, a reatividade do sistema aumenta relativamente ao agente D. Este aumento de reatividade, deve-se ao facto de o agente D assumir posições intermédias nas listas de interação dos agentes A e B (figura 3.2-b). Como resultado, os pares conjugados AD e BD, não serão facilmente destabilizados, e por conseguinte terão tempos de vida superiores do que, por exemplo, os pares AB e AC. Isto porque, os pares AD e BD só podem ser destabilizados por um agente, enquanto que dois agentes podem destabilizar os pares AB e AC.

Se admitirmos que as reações se desencadeiam sempre que uma ligação entre dois agentes dure mais que o tempo mínimo característico, então este aumento nos tempos de vida

permite a existência de reações entre os agentes envolvidos e que não ocorrem entre os agentes AB ou AC, por exemplo.

a)



b)

A	B	C	D
B	C	A	A
D	D	B	B
C	A	-	-

Fig. 3.2- Dinâmica de um sistema frustrado (ABC) na presença de um agente estranho, D. Em a) esquematiza-se a sequência de ligações da dinâmica frustrada. São visíveis dois caminhos possíveis para a destabilização dos pares conjugados. Ambos os caminhos levam à formação do par ligado AD, e consequente ativação da célula T. Em b) apresenta-se a lista de interação para o sistema frustrado (ABC) na presença de um agente estranho, D.

É com recurso a estas propriedades emergentes, que os algoritmos de frustração celular conseguirão fazer discriminação *self-nonself* [9]. No entanto, tal como referido em [9], a aplicação destas ideias ao sistema imunitário, requer algumas modificações na formulação do modelo. Devido à natureza destes sistemas, é necessário considerar modelos com dois tipos de agentes: apresentadores e detetores. A função dos apresentadores será a de apresentar informação do sistema a proteger aos detetores. Uma explicação de porquê que os modelos de frustração celular podem alcançar uma discriminação perfeita entre self e *nonself* é apresentada na secção seguinte.

3.2 Deteção Perfeita por frustração celular

Consideremos um modelo simplificado com dois tipos de agentes, apresentadores e detetores. Cada agente apresentador apresenta uma sequência diferente com a informação que se quer classificar. A população está ainda divida em dois *clusters* de agentes apresentadores e dois *clusters* de agentes detetores como se mostra na Figura 3.3. Todos os agentes apresentadores no mesmo *cluster* possuem os mesmos recetores; todos os agentes detetores no mesmo *cluster* são vistos de forma igual pelos agentes apresentadores.

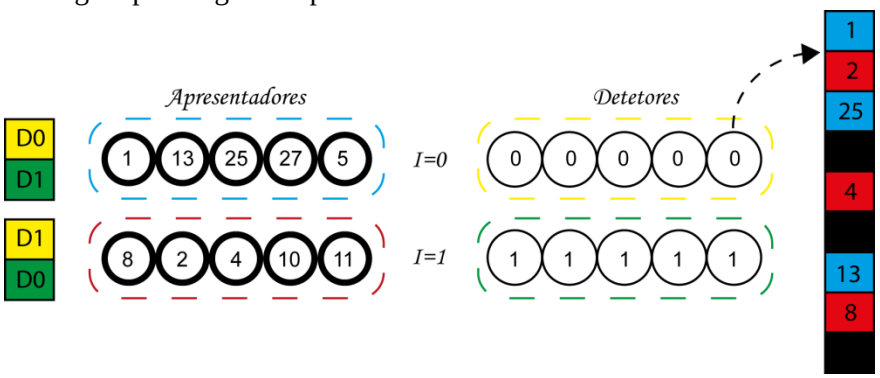


Fig. 3.3-Representação dos agentes (apresentadores e detetores) em *clusters*. De notar as listas de interação dos agentes apresentadores e de um agente detetor. A lista do detetor contem as posições dos agentes do *nonself* representadas a preto.

Os recetores dos detetores permitem ordenar as sequências apresentadas pelos agentes apresentadores em ordens completamente arbitrárias (Figura 3.3). Este é um ingrediente usado em [9] e que foi identificado como sendo essencial durante os trabalhos da tese da minha colega Patrícia Mostardinha. Se as listas de interação puderem ser arbitrariamente diversas e se o sistema imunitário incluir um processo de seleção (designado de seleção negativa) que tenda a ordenar as listas de interação dos agentes detetores de forma a que evitem estabelecer ligações estáveis, então será possível, no final do processo de seleção ordenar as listas como se indica na Figura 3.4. Dado que todos os agentes apresentadores têm no topo da sua lista os agentes detetores do primeiro *cluster*, então todos os agentes detetores desse *cluster* que tenham conseguido sobreviver ao processo de seleção tenderão a ter as sequências apresentadas pelos agentes apresentadores do primeiro *cluster*, nas últimas posições das suas listas de interação.

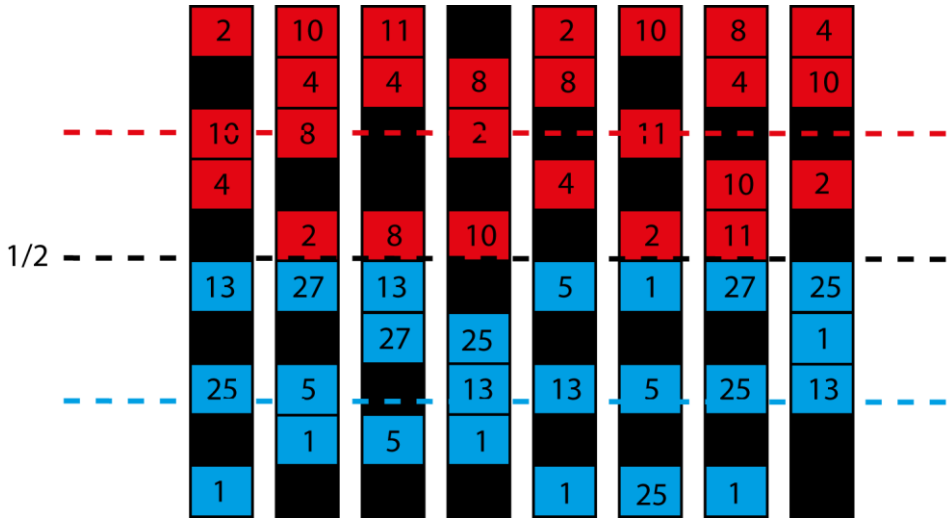


Fig. 3.4-Ilustração das listas de um conjunto de agentes detetores do primeiro cluster. De notar que em média o agente estranho (a preto) se situa a meio da lista de interação, e que os agentes apresentadores do *cluster* 2 se situam no topo da lista de interação.

O processo de seleção negativa é um processo não direcionado, dado que à partida se desconhece que sequências devem estar num ou noutro *cluster*, ou sequer se serão apresentadas. Assim, os processos de seleção negativa a considerar deverão simplesmente substituir os agentes que realizem as ligações mais longas por outros aleatórios.

O resultado importante no que concerne a capacidade de discriminar *self* de *nonself*, consiste na constatação de que as sequências nunca apresentadas ao longo deste processo nunca poderem ter sido condicionadas a ficar numa ou noutra posição das listas de interação. Assim, e contrariamente ao que acontece com as sequências presentes durante o processo de seleção, podemos assumir que as sequências estranhas estarão aleatoriamente distribuídas pelas listas dos agentes detetores. Como resultado, uma fração fixa de agentes detetores do primeiro *cluster* terá sequências estranhas que serão apresentadas por agentes apresentadores do primeiro *cluster* no topo das suas listas, o que provocará tempos de ligação longos e que poderão ser detetados e assinalar a ocorrência de uma intrusão.

Estes argumentos podem ser colocados numa formulação mais matemática. Para tal usamos a descrição matemática apresentada em [9], neste modelo de dois *cluster*. Definimos como N_{A_i} e N_{D_i} o número de agentes apresentadores e detetores do *cluster i*, e as respetivas frequências como $n_{A_i} = \frac{N_{A_i}}{N_A}$ e $n_{D_i} = \frac{N_{D_i}}{N_D}$, onde $N_A = \sum_i N_{A_i}$, $N_D = \sum_i N_{D_i}$ são os números totais de agentes de cada tipo. Representamos também como $N_{A_iD_j}$ e $n_{A_iD_j}$ os números e

frequências de pares onde um agente apresentador do *cluster i* está ligado a um agente detetor do *cluster j*. Finalmente por $N_{A_i\phi}$ e $N_{D_i\phi}$ ($n_{A_i\phi}$ e $n_{D_i\phi}$) designamos os número (frequência) de agentes não ligados.

Podemos então escrever equações de campo médio para a evolução das frequências de pares ligados. As equações contêm dois termos, o primeiro contabiliza todos os termos de criação de pares e o segundo a sua destruição:

$$\frac{dn_{A_iD_j}}{dt} = \Delta(\text{criação } A_1D_1) - \Delta(\text{destruição } A_1D_1) \tag{3.1}$$

A criação de um par A_iD_j ocorre sempre que, por exemplo, agentes não ligados se encontrarem. Da mesma forma, um par A_iD_j é destruído sempre que um dos agentes interagir com um outro que esteja mais bem colocado na sua lista e o mesmo ocorra com esse agente. Todos os termos devem ser considerados. Em particular, para um par A_1D_1 temos as seguintes taxas de criação e destruição:

$$\Delta(\text{criação } A_1D_1) = \mathrm{n}_{\mathrm{A}_1\phi}\left(\mathrm{n}_{\mathrm{D}_1\phi} + \frac{1}{2}\mathrm{n}_{\mathrm{A}_1D_1}\right) + \mathrm{n}_{\mathrm{A}_1D_2}\left(\mathrm{n}_{\mathrm{D}_1\phi} + \frac{1}{2}\,\mathrm{n}_{\mathrm{A}_1D_1}\right) \tag{3.2}$$

$$\Delta(\text{destruição } A_1D_1) = \mathrm{n}_{\mathrm{A}_1D_1}\left(\mathrm{n}_{\mathrm{A}_2\phi} + \frac{1}{2}\mathrm{n}_{\mathrm{A}_1\phi} + \frac{1}{2}\mathrm{n}_{\mathrm{A}_1D_2}\right) \tag{3.3}$$

Nestas equações assumi que o sistema estava perfeitamente educado [9, 10]. Consequentemente, por exemplo, quando o agente D_1, ligado a um agente A_1, interagir com um outro agente apresentador A_1 do *cluster* 1, o último terá uma probabilidade de ½ de estar mais bem colocado na sua lista de preferência, o que levaria a que o agente detetor tenha essa probabilidade de querer mudar de par. As equações completas são:

$$\frac{dn_{A_1D_1}}{dt} = \mathrm{n}_{\mathrm{A}_1\phi}\mathrm{n}_{\mathrm{D}_1\phi} + \mathrm{n}_{\mathrm{A}_1D_2}\mathrm{n}_{\mathrm{D}_1\phi} - \mathrm{n}_{\mathrm{A}_1D_1}\mathrm{n}_{\mathrm{A}_2\phi} \tag{3.4}$$

$$\frac{dn_{A_1D_2}}{dt} = \mathrm{n}_{\mathrm{A}_1\phi}\mathrm{n}_{\mathrm{D}_2\phi} + \mathrm{n}_{\mathrm{A}_1\phi}\mathrm{n}_{A_2\mathrm{D}_2} - \mathrm{n}_{\mathrm{A}_1D_2}\left(\mathrm{n}_{\mathrm{D}_1\phi} + \frac{1}{2}\mathrm{n}_{\mathrm{A}_1D_1}\right) \tag{3.5}$$

$$\frac{dn_{A_1\phi}}{dt} = \mathrm{n}_{\mathrm{A}_1D_1}\left(\mathrm{n}_{\mathrm{A}_2\phi} + \frac{1}{2}\mathrm{n}_{\mathrm{A}_1D_2}\right) - \mathrm{n}_{\mathrm{A}_1\phi}\left(\mathrm{n}_{\mathrm{D}_1\phi} + \mathrm{n}_{\mathrm{D}_2\phi} + \mathrm{n}_{\mathrm{A}_2D_2}\right) \tag{3.6}$$

$$\frac{dn_{D_1\phi}}{dt} = \mathrm{n}_{\mathrm{A}_2D_1}\left(\mathrm{n}_{\mathrm{D}_2\phi} + \frac{1}{2}\mathrm{n}_{\mathrm{A}_2D_2}\right) - \mathrm{n}_{\mathrm{D}_1\phi}\left(\mathrm{n}_{\mathrm{A}_1\phi} + \mathrm{n}_{\mathrm{A}_2\phi} + \mathrm{n}_{\mathrm{A}_1D_2}\right) \tag{3.7}$$

E de forma semelhante para as equações que envolvem A_2D_1, A_2D_2 e $A_2\phi$. Na presente análise interessa-nos particularmente determinar os tempos de ligação dos pares de agentes. Ora analisando os termos de destruição, podemos obter as taxas de destruição de pares, e assim os inversos dos seus tempos de vida: $\Delta(\text{destruição } A_1D_1) = \mathrm{n}_{\mathrm{A}_1D_1}\left(\mathrm{n}^{\mathrm{eq}}_{A_2\phi} + \frac{1}{2}\mathrm{n}^{\mathrm{eq}}_{A_1\phi} + \frac{1}{2}\mathrm{n}^{\mathrm{eq}}_{A_1D_2}\right) = \mathrm{n}_{\mathrm{A}_1D_1}\,\tau^{-1}_{A_1D_1}$. Nesta equação as frequências são medidas no equilíbrio. As expressões para os diversos tempos de vida e para os diversos pares são então:

$$\tau^{-1}_{A_1D_1} = \mathrm{n}^{\mathrm{eq}}_{A_2\phi} + \frac{1}{2}\mathrm{n}^{\mathrm{eq}}_{A_1\phi} + \frac{1}{2}\mathrm{n}^{\mathrm{eq}}_{A_1D_2} \tag{3.8}$$

$$\tau_{A_1D_2}^{-1} = \mathrm{n}_{D_1\phi}^{\mathrm{eq}} + \frac{1}{2}\mathrm{n}_{A_1\phi}^{\mathrm{eq}} + \frac{1}{2}\mathrm{n}_{A_1D_1}^{\mathrm{eq}} \tag{3.9}$$

e de forma semelhante para os $\tau_{A_2D_1}^{-1}$ e $\tau_{A_2D_2}^{-1}$.

Importa agora considerar o tempo de vida de uma sequência estranha apresentada por um agente do *cluster* 1 e ligado a um detetor do *cluster* 1. Neste caso em média metade dos agentes apresentadores colocá-lo-ão na primeira metade da sua lista de interação. Assim isso implica que para o agente apresentando a sequência estranha, A_P, teremos:

$$\tau_{A_PD_1}^{-1} = \frac{1}{2}\mathrm{n}_{A_2\phi}^{\mathrm{eq}} \tag{3.10}$$

A taxa com que estes pares podem ser destruídos é muito baixa, pois só metade dos agentes apresentadores do segundo *cluster* os poderá destabilizar. Consequentemente o tempo de duração destes pares será muito longo e poderá assim ser detetado. Resolvendo numericamente as equações (3.4-3.7) e calculando o rácio entre os tempos de vida usando:

$$\frac{\tau_{A_PD_1}}{\tau_{A_1D_1}} = \frac{2\mathrm{n}_{A_2\phi}^{\mathrm{eq}} + \mathrm{n}_{A_1\phi}^{\mathrm{eq}} + \mathrm{n}_{A_1D_2}^{\mathrm{eq}}}{\mathrm{n}_{A_2\phi}^{\mathrm{eq}}} \tag{3.11}$$

Obtemos os resultados da figura 3.5. O rácio, apesar de claramente maior que 1, pode ser, no entanto, muito maior numa situação prática. Isto porque fizemos a assunção de que o par envolvendo o agente apresentador com a sequência estranha seria destabilizado por metade dos agentes do *cluster* 2. Ora numa situação prática esse seria o pior cenário. Na realidade o agente detetor pode mesmo colocar a sequência estranha no topo da sua lista de interação, situação que levaria a que nenhuma interação conseguisse destabilizar o par envolvendo a sequência estanha. Nesse caso o denominador da expressão (3.11) tenderia para zero e o rácio divergia.

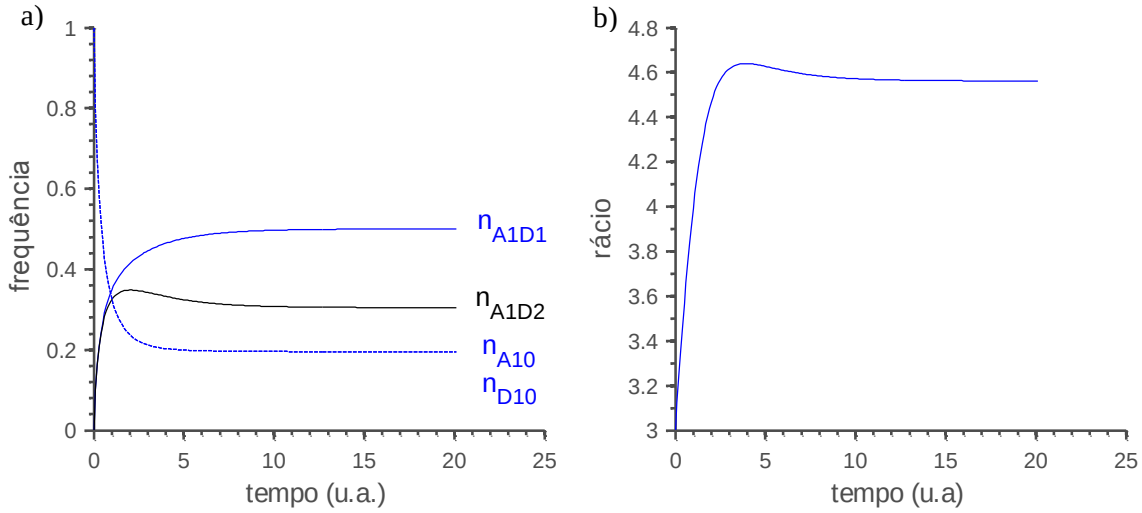


Fig. 3.5-Em a) a evolução das frequências dos conjugados, A_1D_1 e A_1D_2, e dos agentes não conjugados A_{10} e D_{10}. Em b) a evolução do rácio.

Ainda que os argumentos anteriores possam parecer convincentes, não respondem a uma questão quantitativa muito importante: quantas sequências estranhas conseguirão evitar a deteção? Esta questão requer uma abordagem computacional pois não pode ser respondida com total certeza com argumentos analíticos, que em geral contêm aproximações (tal como a assunção de "campo médio") e por isso não conseguem contemplar toda a diversidade do problema. Em particular, queremos também saber se as flutuações estocásticas inerentes a estes problemas podem ter um efeito importante sobre a capacidade de deteção de intrusões. Por fim

não devemos esquecer que o argumento anterior assumiu que a educação tinha conseguido ordenar as listas de forma perfeita. Ora esse não é com certeza o caso de sistemas onde as listas são compostas de inúmeros elementos. Além disso, essa ordenação tem de ser realizada de forma não dirigida, ou seja, trocando simplesmente os agentes detetores por outros com listas aleatórias.

3.3 Modelo de frustração celular

De forma a tornar a apresentação do modelo clara e concisa, apresentar-se-á um conjunto de definições. A primeira definição, e provavelmente a mais importante é a definição de agente. O agente pode ser considerado como o bloco fundamental do modelo, e é definido da seguinte forma:

> ***Definição 3.3.1 (Agente):*** Um agente $a_i \in \boldsymbol{A}$ pode ser definido por uma sequência ordenada $a_i = \{s_i, l_i, R_i, p_i, C_i\}$, onde:
>
> - $s_i \in \Sigma^n$ é uma sequência de n dígitos binários, $\Sigma^n = \{b_1, b_2, \dots, b_n\} \wedge b = \{0,1\}$, denotada por *string* ou ligando.
> - l_i é uma sequência de todas as *strings* contidas em Σ^n, e é denominada de lista ou receptor.
> - R_i é o conjunto das regras de interação.
> - $p_i \in \mathbb{N}_0$ específica o estado de conjugação do agente, detendo o índice do agente com o qual está conjugado, ou zero se estiver sozinho.
> - C_i é o conjunto de todos os agentes com os quais o agente a_i pode interagir.

Dada a natureza imunológica do modelo decidiu-se considerar dois tipos de agentes, apresentadores e detetores. A consideração destes dois tipos de agentes resulta da observação do sistema imunitário. No sistema imunitário as células apresentadoras (APCs) varrem o corpo à procura de antigénios. Ao encontrarem, dirigem-se para os nós linfáticos, onde os apresentam às células T. As células T agem então como detetores, acionando uma resposta imunitária se reconhecerem antigénios estranhos (*nonself*). No nosso caso, os agentes apresentadores podem apresentar aos detetores uma qualquer *string* contida em Σ^n, o que dá lugar à seguinte definição:

> ***Definição 3.3.2 (Tipos de agentes):*** O conjunto de agentes $\boldsymbol{A}$ é formado por dois grupos de agentes: apresentadores, A, e detectores, D, de tal forma que $\boldsymbol{A} = A \cup D$.

De forma a manter o modelo simples, é assumido que agentes do mesmo tipo não podem interagir. Esta assunção permite definir a conectividade de cada agente, a_i, como:

- $\forall\, i \in D: C_i = A$
- $\forall\, i \in A: C_i = D$

O estado de conjugação, p_i, na definição 3.3.1, assume que num determinado instante um agente pode fazer no máximo um contacto, como referido anteriormente. A definição 3.3.1, comporta ainda um conjunto de regras de interação, R_i, para cada agente. Este conjunto de regras tem por base as decisões de cada agente e governa a formação de pares. A decisão feita por um determinado agente, a_i, leva em conta como as *strings* apresentadas pelos outros agentes são classificadas na sua lista, l_i. A classificação de uma determinada *string*, s, numa lista, l, é definida como $K_l(s)$. Isto permite definir o seguinte conjunto de regras:

Definição 3.3.3 (regras de interação): O seguinte conjunto de regras, define como atualizar os estados dos agentes, quando os agentes a_i e a_j interagem:

- Se $p_i = p_j = 0 \rightarrow p_i = j, p_j = i$
- Se $p_i = n \wedge p_j = 0 \wedge K_{l_i}(s_j) < K_{l_i}(s_n) \rightarrow p_i = j, p_j = i, p_n = 0$
- Se $p_i = 0 \wedge p_j = n \wedge K_{l_j}(s_i) < K_{l_j}(s_n) \rightarrow p_i = j, p_j = i, p_n = 0$
- Se $p_i = m \wedge p_j = n \wedge K_{l_i}(s_j) < K_{l_i}(s_m) \wedge K_{l_j}(s_i) < K_{l_j}(s_n) \rightarrow p_i = j, p_j = i, p_m = 0, p_n = 0$

No entanto no decorrer deste trabalho verificou-se ser vantajosa a organização dos agentes apresentadores e detetores em *clusters (ou grupos)*. São ainda consideradas um conjunto de assunções simplificadoras. No caso dos agentes apresentadores considera-se que, os agentes apresentadores pertencentes ao mesmo *cluster* contêm os mesmos recetores. Por sua vez, os detetores pertencentes a um mesmo *cluster*, apresentam a mesma *string*. Estas assunções não afetam a generalidade do modelo, pois as *strings* apresentadas pelos apresentadores são tão diversas quanto se desejar.

A organização tanto de apresentadores como detetores, em *clusters*, é feita em c conjuntos disjuntos. Desta forma, os detetores pertencentes ao *cluster* $I, \forall\, I \in \{0, \dots, c-1\}$, apresentam a *string* $s = I$. Os apresentadores, tem uma lista, l, na qual a posição n é descrita como: $l(n) = (n - I - 1) mod\ c$.Uma representação deste modelo encontra-se ilustrada na figura 3.6.

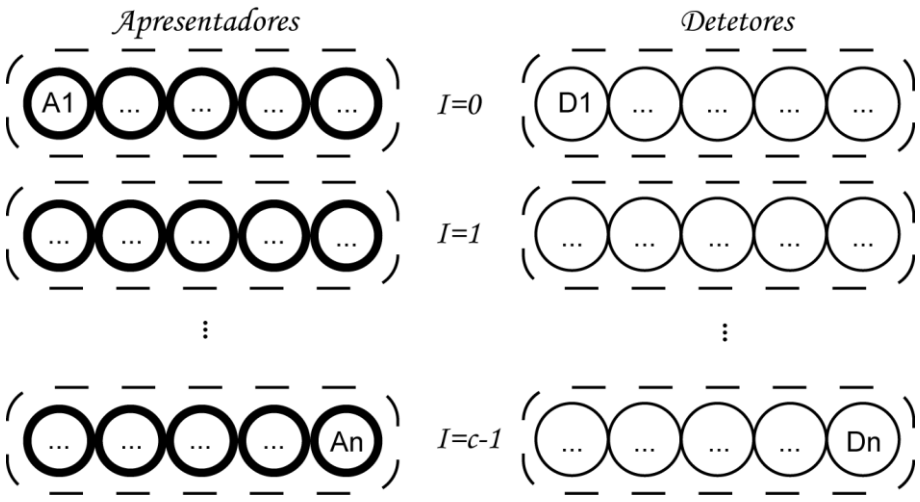


Fig. 3.6- Ilustração do modelo com organização em *clusters* estudado neste trabalho. Á esquerda representa-se os agentes apresentadores e à direita, os agentes detetores. Os elementos pertencentes a um mesmo *cluster* encontram-se destacados por uma linha a tracejado. De realçar que os agentes apresentadores usam a informação do número do *cluster* a que pertencem para classificar os detetores. Por sua vez a sequência exibida pelos detetores, é igual ao valor do *cluster* a que pertencem.

Cada *cluster I* pode conter um número variável de agentes, no entanto a descrição aqui efetuada vai considerar apenas modelos com clusters simétricos. Isto significa que, um determinado *cluster* de apresentadores, contêm o mesmo número de agentes que o correspondente *cluster* de detetores. Se denotarmos o número de agentes em cada *cluster*, de apresentadores e detetores, por N_{A_I} e N_{D_I} respetivamente, então assume-se que $N_{A_I} = N_{D_I}$. O número total de agentes apresentadores e detetores será denotado por N_A e N_D e o número total de agentes, por N.

Após efetuada a descrição dos constituintes básicos do modelo, importa agora estabelecer como se aplicam as regras de interação durante uma iteração. Durante uma iteração, todos os agentes de um determinado tipo são colocados em interação com uma sequência aleatória de agentes do outro tipo. Em cada interação, as regras de interação (*Definição **3.3.3***), são usadas para atualizar os tempos dos pares conjugados e estabelecer novas configurações. Uma configuração é definida como:

> ***Definição 3.3.4 (Configuração):*** Uma configuração dos sistemas frustrados celulares, L, pode ser definida como o conjunto de agentes não conjugados mais o conjunto de agentes conjugados:
>
> $$L = \{a_i : p_i = 0, i = 1, \dots, N\} \cup \{(a_i, a_j) : p_i = j \wedge p_j = i, i, j = 1, \dots, N\}$$

Note-se que durante uma iteração estocástica vários agentes podem interagir com um mesmo agente, o que pode requerer definir uma sequência de eventos dentro de uma mesma configuração. No entanto, do ponto de vista deste trabalho, a sua definição não é relevante pois o modelo apenas se preocupa com as propriedades que emergem da sequência das configurações definidas para cada iteração.

Um outro conceito importante é o do tempo de vida dos pares conjugados que podem ser definidos como:

> ***Definição 3.3.5 (Tempos de vida dos pares conjugados):*** O tempo de vida de um par conjugado, $(a_i \ , \ a_j)$, é o número de iterações decorridas, entre duas alterações consecutivas dos estados de conjugação, p_i e p_j.

Para obter boas discriminações *self-nonself*, interessa que estes tempos de vida sejam mínimos na ausência de elementos estranhos. Com vista à sua minimização, o modelo requer a aplicação de uma etapa de educação do repertório de agentes detetores. A etapa de educação consiste na seleção das listas dos detetores, atuando como um tipo de seleção negativa que pode ser definido como:

> ***Definição 3.3.6 (Seleção negativa):*** A seleção negativa troca os recetores dos agentes detetores que participem nas ligações mais longas em $W \in \mathbb{N}$ iterações, por outros aleatórios.

A etapa de educação seleciona os detetores que formam pares conjugados com pequenos tempos de vida com agentes apresentando sequências do *self*. No entanto, de forma a assegurar que isto apenas acontece com o repertório do *self*, o modelo considera uma etapa de monitorização. Esta etapa, consiste em apresentar elementos não pertencentes ao repertório do *self*, (i.e. elementos estranhos - *nonself*), aos detetores e monitorização dos tempos de vida dos pares conjugados. A etapa de monitorização pode ser definida como:

> ***Definição 3.3.7 (Monitorização ou deteção):*** A etapa de monitorização analisa a frequência com que ligações dos agentes apresentadores durante $W \in \mathbb{N}$ iterações com uma duração pré-definida são realizadas. Assinala uma anomalia ou intrusão quando estes valores excederem os valores registados na fase da educação.

Na prática, neste trabalho, utilizaremos a fase de monitorização para apresentar uma amostra onde só uma sequência foi alterada num agente apresentador. O objetivo será o de estudar quantas sequências estranhas poderão não ser detetadas pelo sistema e comparar o desempenho deste sistema de deteção com o proposto pelos ASN.

A descrição do modelo aqui considerada leva em conta que a conectividade dos agentes é total. Isto é, um agente de um determinado pode tipo interagir com todos os agentes do outro tipo. No entanto, pode ser vantajoso considerar conectividades mais pequenas, usando subconjuntos de As e Ds, assim como um número de agentes por *cluster* variável. O estudo destas características foge aos objetivos deste trabalho, pelo que se deixa como trabalho futuro. Na secção a seguir, irão ser discutidos métodos para comprimir a informação armazenada na lista de interação, l, de cada agente.

3.4 Listas de interação implícitas

De seguida apresentaremos as várias regras de codificação, como se aplicam e quantas listas diferentes conseguem gerar. De notar que o número total de listas que pode ser gerado cresce de forma combinatória. Com *p* bits podemos ter 2^p *strings* e consequentemente $2^p!$ listas diferentes. Como se verá, o número de listas diferentes que um algoritmo de codificação consegue gerar terá impacto na qualidade da deteção do algoritmo, pelo que a definição de algoritmos que consigam compactar essa informação será muito importante, tendo em vista aplicações práticas com um número grande de elementos (p grande). A ideia base de funcionamento dos algoritmos de codificação das listas de interação consiste em aplicar sobre uma sequência de dígitos binários, um conjunto de operações, calculando uma pontuação. Ao aplicar o algoritmo a todo o conjunto de sequências binárias com *n* bits atribuir-se-ão pontuações a cada sequência, que poderão ser usadas para ordenar as sequências numa ordem desejavelmente arbitrária.

Definição 3.4.1 (Regra de Codificação de Listas de Interação): Uma regra de codificação é uma função *f* que aplicada sobre o conjunto de números naturais $\{1,\ldots, 2^n\}$ produz uma sequência $(b_1, b_2, \ldots, b_{2^n})$, com $(b_1, b_2, \ldots, b_{2^n}) \in \{1, \ldots, 2^n\}$.

3.4.1 TWR

A regra *TWR* opera um conjunto de operações binárias sobre a *string s* com *n* dígitos binários e que codifica a informação contida no ligando em interação. A primeira operação a ser efetuada é a operação *Toggle*. Esta operação corresponde à aplicação do ou-exclusivo (XOR) entre a *string s* e uma sequência *T*. Depois aplica-se a operação $(\,)_{Weights}$, que atribui diferentes pesos a cada um dos dígitos binários resultantes da operação anterior. Esta operação é equivalente a uma permutação da sequência binária. Finalmente, subtrai-se ao valor correspondendo à codificação binária obtida, uma referência R. Uma ilustração deste cálculo pode ser encontrada na figura 3.7. Sinteticamente podemos escrever esta regra na forma:

$$K_l(s) = (T \oplus s)_W - R \tag{3.12}$$

De forma a determinar o número de listas diferentes obtidas por esta regra, geraram-se todas as combinações que é possível obter com os diferentes valores de *W*, *T* e *R* para um determinado número de bits, *n*. Calcularam-se todas as pontuações para todas as *s strings* de *n* bits e ordenaram-se as *s strings* de acordo com essas pontuações. O resultado dessa ordenação

constitui uma lista. Implementou-se um algoritmo que determinou quantas listas não repetidas foram geradas usando este procedimento. O resultado é apresentado na tabela 3.1

Tabela 3.1-Número de listas diferentes obtidas com a regra *TWR*

***n* bits**	2	3	4	5	6
nº de listas possíveis $(2^n!)$	24	4.03×10^4	2.09×10^{13}	2.63×10^{35}	1.27×10^{89}
nº de listas diferentes obtidas com a regra *TWR*	16	1.92×10^2	3.03×10^3	6.14×10^4	1.47×10^6
$\frac{Listas_{TWR}}{2^n!}$ **(%)**	67	0. 47	10^{-8}	10^{-29}	10^{-81}

Com *n=2* a fração de listas geradas com a regra *TWR* relativamente ao número máximo possível é de 66.67%. No entanto, esta fração decresce rapidamente. Para *n=3* este valor desce para menos de 1%, e muito rapidamente depois. No entanto note-se que o número de listas que se consegue obter mesmo com 6 bits é mesmo assim considerável: 10^6. Assim, apesar de simples, esta regra permite criar uma grande diversidade de listas. O custo é pequeno: a regra requer só dois números binários (T e R) e uma sequência de pesos.

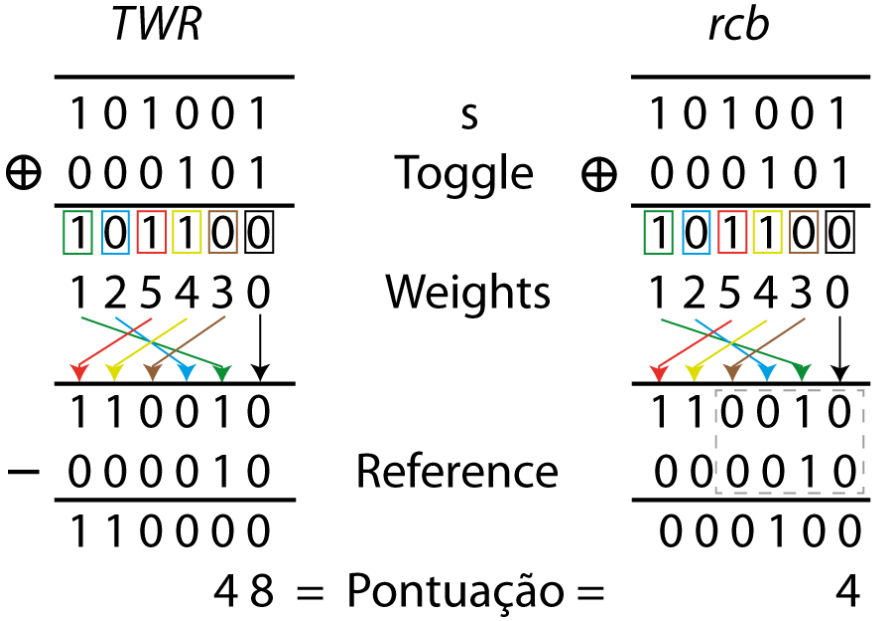


Fig. 3.7-Ilustração do cálculo da pontuação usando a regra *TWR* (à esquerda) e a regra *#rcb* (à direita).

3.4.2 #rcb

A regra *#rcb* foi inspirada na métrica de *r-contiguous bits* (proposta para modelos em imunologia em [11]), e na regra anterior. Difere da última regra somente na última operação. Em vez da operação diferença, contabiliza-se o maior número dígitos binários iguais e consecutivo. A ideia desta regra é a de quantificar a semelhança entre duas sequências binárias. Sinteticamente podemos definir esta regra na forma:

$$K_l(s) = f_{\#rcb}((T \oplus s)_W, R) \tag{3.13}$$

Uma característica desta regra é a elevada degenerescência nas pontuações, i.e., a existência de um elevado número de sequências que produzem o mesmo resultado. Esta repetição verifica-se porque é possível construir muitas *strings* contendo o mesmo número máximo de bits contíguos, *r*, mas diferindo nos restantes. Esta degenerescência na classificação reflete-se no número de listas diferentes que é possível obter com a regra *#rcb* (tabela 3.2).

Tabela 3.2-Número de listas obtidas com a regra #*rcb*

n **bits**	2	3	4	5	6
nº de listas possíveis ($2^n!$)	24	$4.03\text{x}10^4$	$2.09\text{x}10^{13}$	$2.63\text{x}10^{35}$	$1.27\text{x}10^{89}$
nº de listas diferentes obtidas com a regra #*rcb*	4	24	$1.92\text{x}10^2$	$1.92\text{x}10^3$	$2.31\text{x}10^4$
$\frac{Listas_{\#rcb}}{2^n!}$ **(%)**	17	0. 06	10^{-10}	10^{-31}	10^{-83}

Para um número de bits *n*=2, a regra *#rcb* gera 17% das listas possíveis. Quando o número de bits aumenta de 2 para 3 o número de listas geradas é inferior a 0.1%. Esta divergência entre o número máximo de listas que se poderão gerar e o número de listas gerado pela regra está muito clara na tabela 3.2. Note-se no entanto que, à semelhança do que sucedia com a regra *TWR*, aumentando o número de bits o número de listas geradas cresce exponencialmente. Assim coloca-se a questão de saber se, apesar destas regras gerarem somente uma parte muito reduzida da diversidade acessível, se essa diversidade não será suficiente para os propósitos do desenvolvimento de um sistema imunitário baseado na frustração celular.

3.4.3 Regras aleatórias

Dada a limitação identificada na capacidade em gerar toda a diversidade acessível com as regras *TWR* e *#rcb*, estabeleceu-se como um dos propósitos desta tese desenvolver regras mais eficazes. Pensou-se em estabelecer regras que recorram a geradores de números pseudoaleatórios. Ao contrário das regras anteriores, a motivação que as originou não partiu da imunologia. O objetivo foi o de elaborar regras potentes suscetíveis de aplicação em algoritmos de inteligência artificial. O outro objetivo, claro está, foi o de verificar se usando regras mais eficientes, o desempenho do algoritmo poderia melhorar.

Os processos de geração de números pseudoaleatórios podem ser muito variados, como se pode verificar na referência [12]. No entanto, em quase todos os métodos define-se um processo iterativo começando num estado Y_0:

$$Y_{j+1} = f_1(Y_j) \tag{3.14}$$

Posteriormente pode ainda ser definido um processo de saída, para gerar um número aleatório x_j final, uniformemente distribuído no domínio pretendido:

$$x_j = g(Y_j) \tag{3.15}$$

A primeira regra de codificação baseada em geradores pseudoaleatórios que aqui apresentamos designamos por G^0G^1. Esta regra recorre à aplicação sequencial e alternada de dois geradores de números pseudoaleatórios, G^0 ou G^1, sobre um estado inicial $K_l^0(s)$, definido

a partir de uma sequência binária, *l*, que define o recetor. A aplicação de um ou outro gerador é determinada pelos bits que compõem a *string s* (ver figura 3.8). Verificou-se que se obtinham melhores resultados se, a cada iteração, o gerador fosse aplicado duas vezes em vez de uma. O algoritmo pode ser escrito formalmente da seguinte forma:

$$K_l^{i+1}(s) = G^1\left(K_l^i(s)\right)^2 \delta_{u_i,1} + G^0\left(K_l^i(s)\right)^2 \delta_{u_i,0} \tag{3.16}$$

onde, u_i representa o bit *i* da *string s* (*i*=0,1,…, *n)*. O bit u_i que determina a evolução do algoritmo pode ser obtido através da relação $u_i = \left\lfloor \frac{s}{2^i} \right\rfloor \cdot 1$, onde $(\cdot)$ representa a instrução lógica AND, e $\lfloor . \rfloor$, a truncatura para inteiros.

Para a implementação desta regra de codificação usaram-se dois geradores lineares congruentes (GLCs). Neste tipo de gerador, cada número determina o seu sucessor por meio de uma função linear simples, seguida de uma redução modular. A forma base de um GLC aplicada ao problema em questão é:

$$G^j\left(K_l^i(s)\right) = \left(a_j G^j\left(K_l^i(s)\right) + c_j\right) \, mod \, m_j, \qquad j = 0{,}1 \tag{3.17}$$

$$0 \leq Y^j < m_j \tag{3.18}$$

onde a_j é denominado de multiplicador, c_j de incremento e m_j de módulo. Os parâmetros de a_j, c_j e m_j escolhidos para este trabalho foram:

$$m_0 = m_1 = 2^{32} \tag{3.19}$$

$$a_0 = 214013, \;\; a_1 = 134775813 \tag{3.20}$$

$$c_0 = 2531011, \; c_1 = 1 \tag{3.21}$$

Estes parâmetros foram escolhidos por possuírem o mesmo período *m* e por serem bastante usados em compiladores como o Microsoft Visual Studio, o Borland Delphi e o Virtual Pascal.

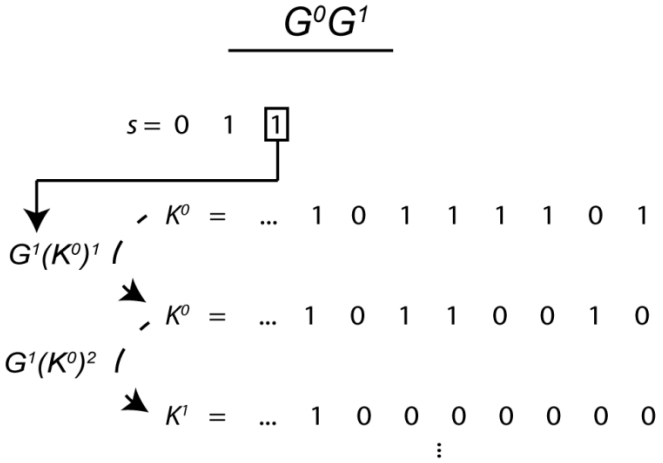


Fig. 3.8- Ilustração de um passo no cálculo da classificação para a regra G^0G^1

Através deste algoritmo conseguiu-se associar a cada palavra de *n* bits, um número aleatório. Este número aleatório foi utilizado para ordenar as diferentes palavras em listas. O número de listas diferentes geradas usando este algoritmo está apresentado na Tabela 3.3. O

comportamento do algoritmo revelou-se perfeito, pelo menos para espaços de sequências até 3 bits, ou seja 8 elementos. Para n>3 a análise do número de listas diferentes geradas é computacionalmente inviável dado o seu número: $2^4! \sim 10^{13}$. Alternativamente, optou-se por confirmar que se conseguiriam obter todas as listas com 10 elementos. Efetivamente, obtiveram-se todas as $10! = 3628800$ ordenações possíveis, o que permite supor que o mesmo sucederá em espaços de dimensão superior.

Tabela 3.3-Número de listas diferentes obtidas com a regra G^0G^1. A obtenção das listas só é possível para um número de bits inferior a 3, visto que para um número de bits igual ou superior a 4 o armazenamento das listas na memória RAM do computador é inviável.

n **bits**	2	3
nº de listas possíveis ($2^n!$)	24	4.03×10^4
nº de listas diferentes obtidas com a regra G^0G^1	24	4.03×10^4
% $\frac{Listas_{G^0G^1}}{2^n!}$	100%	100%

A segunda regra a ser aqui discutida foi designada por *Y*, e segue uma abordagem um pouco diferente. Ao invés de recorrer à aplicação de dois geradores para obter as listas, esta regra inspirou-se no gerador de números aleatórios de Tausworthe [12]. Este gerador funciona de uma forma semelhante ao de uma máquina de Turing sobre uma sequência de dados, conforme se ilustra na Figura 3.9.

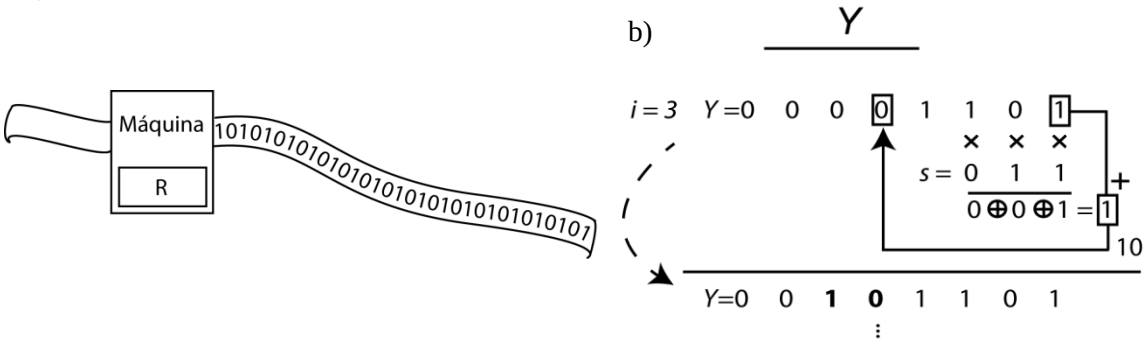


Fig. 3.9-Em a) a ilustração da máquina de turing, e em b) a ilustração do seu funcionamento.

Para obter um novo número aleatório *Y* com *n* bits, a máquina opera sobre o número anterior *b* vezes, onde $b \geq n$. De cada vez, aplica sobre a fita uma operação que depende de uma sequência característica da máquina (vide recetor) e da sequência *s* do ligando. Em cada operação é calculado um bit, cujo resultado é incorporado na fita para ser utilizado na operação seguinte. Formalmente a operação para determinar o bit *i* da sequência *Y*, pode ser escrita da seguinte forma:

$$Y_i \equiv (s_n Y_{i-n} \oplus s_{n-1} Y_{i-n+1} \oplus \dots \oplus s_1 Y_{i-1}) \qquad (3.22)$$

Aqui a adição $\oplus$ representa a operação ou-exclusivo (XOR). A aplicação da equação requer que, para calcular o bit *n+1*, a fita contenha pelo menos *n* bits. Um novo número aleatório *Y* com *n* bits é obtido somente ao fim *b* iterações desta equação.

Verificámos que a equação (3.22) atribuía sempre ao elemento zero a mesma pontuação, quando a sequência *s* fosse nula. Isto implicaria que haveria um elemento nas listas

que ficaria sempre na mesma posição. Ou seja, as listas não seriam realmente aleatórias. Para evitar este problema, sem ter que impor uma restrição aos valores de *s* possíveis, alterou-se a expressão 3.22 para:

$$Y_i \equiv Y_{i-n} + (s_n Y_{i-n} \oplus s_{n-1} Y_{i-n+1} \oplus \ldots \oplus s_1 Y_{i-1}) \qquad (3.23)$$

que garante que o zero deixe de ser um ponto fixo da fórmula de recorrência. Confirmou-se que efetivamente esta regra reproduz todas as listas que seria possível definir com 2^n elementos, conforme se pode verificar na tabela 3.4.

Tabela 3.4-Número de listas diferentes obtidas com a regra *Y*. A obtenção das listas só é possível para um número de bits inferior a 3, visto que para um número de bits igual ou superior a 4 o armazenamento das listas na memória RAM do computador é inviável.

n **bits**	2	3
nº de listas possíveis ($2^n!$)	24	4.03×10^4
nº de listas diferentes obtidas com a regra *Y*	24	4.03×10^4
$\% \frac{Listas_Y}{N!}$	100%	100%

3.4.4 Considerações finais

Nesta secção foram descritas quatro regras para a definição implícita das listas de interação. As regras *#rcb* e *TWR* têm a vantagem de serem as mais plausíveis de um ponto de vista imunológico. Definem a lista associada a um recetor através da definição de um conjunto de dados *l={ T, W, R}* e uma operação. Os dados são compostos por dois números, *T* e *R*, e uma sequência *W* de pesos em número igual ao número de bits da sequência apresentada. A operação é uma operação aritmética (no caso da regra *TWR*) ou binária (no caso da contagem de bits contíguos na regra *#rcb*).

As regras de codificação baseadas em geradores de números aleatórios foram aqui apresentadas pela primeira vez. A regra G^0G^1 usa dois geradores aleatórios para criar uma pontuação dependente da sequência apresentada, a qual funciona como um número semente. A sequência de bits que compõe a sequência determina qual o gerador a ser aplicado. A regra *Y* consiste numa alteração ao método de geração de números aleatórios de Tausworthe.

Ambas as regras baseadas em listas aleatórias conseguiram produzir a totalidade das listas que deveria ser possível definir com 2^n elementos. No entanto, não deixam de ser métodos de geração de números pseudoaleatórios. Assim sendo, as listas que estas regras geram implicitamente podem ter correlações escondidas cujo impacto no algoritmo de frustração celular importa estudar. É fácil de imaginar que muitas outras definições de listas implícitas são possíveis, pelo que este estudo pode estar só no seu começo.

Capítulo 4: Implementação

Muitos dos processos que ocorrem na natureza ocorrem em paralelo. Porém, a sua simulação computacional tem sido efetuada sequencialmente. No entanto, com o aumento da complexidade dos modelos, a simulação sequencial dos processos pode tornar-se demasiado limitativa. A computação paralela surgiu por isso como uma alternativa à computação sequencial tradicional. No entanto e até recentemente, a computação paralela estava limitada ao uso de supercomputadores e *clusters* de computadores. Mas estes, devido ao elevado investimento e custos operacionais estavam apenas acessíveis a grandes instituições.

As unidades de processamento gráfico (conhecidas pela sigla inglesa *GPU*) vieram alterar esta situação, tornando acessível a uma comunidade mais abrangente o poder computacional dos *clusters* de computadores. No entanto, de forma a obter desempenhos comparáveis, o programador tem de incorporar na programação conhecimentos da arquitetura dos *GPUs*. De facto, os *GPUs* só obtêm as melhores performances, quando o algoritmo tira partido de acessos coalescidos à memória, do uso da memória partilhada (*shared memory*) entre programas independentes (*threads)* e do paralelismo de dados [13-15].

Estas considerações são cruciais na implementação do algoritmo. Uma delas é inerente a qualquer paralelização. Consiste na possibilidade de surgirem conflitos de memória resultantes da atualização de uma variável por diferentes *threads*. Estes conflitos potenciais surgem naturalmente no modelo de frustração celular. Além disto, é ainda necessário adotar estratégias de programação que permitam que o algoritmo possa beneficiar de acessos rápidos à memória.

Neste capítulo discutiremos a paralelização do algoritmo de frustração celular. Primeiro descreveremos as dificuldades na paralelização do algoritmo, e como podem ser resolvidas. De seguida, é apresentado o fluxograma do algoritmo final, e descrito o seu funcionamento. Por último é efetuada uma comparação de desempenhos das versões sequencial e paralela.

4.1 Evitando os conflitos nas decisões

A paralelização do algoritmo, tal como apresentada anteriormente não é imediata. Isto deve-se ao facto de ser necessário garantir que, em cada iteração, cada agente efetua no máximo um contacto. Isso não se verifica facilmente em algoritmos paralelizados e pode ter consequências graves na evolução da dinâmica do sistema, pois o estado de cada agente pode deixar de estar bem definido. Na figura 4.1 mostra-se como tal pode suceder se o algoritmo não for desenhado cuidadosamente: um determinado agente a_i pode ficar ligado a dois agentes diferentes na iteração seguinte. Estes conflitos da definição do estado seguinte, surgem se todos os agentes forem tratados de forma igual no sorteio das interações, o que, à primeira vista, poderia parecer o mais natural. Assim, se o agente a_i tentar formar um conjugado com o agente a_k, e o agente a_j tentar simultaneamente formar um conjugado com o agente a_i, a aplicação independente das regras de interação, pode associar o agente a_i a dois pares diferentes na configuração seguinte (figura-4.1-a). Um outro exemplo deste tipo de conflitos surge quando dois agentes conjugados interagem independentemente com outros dois agentes. Se uma destas interações levar a uma alteração no estado de conjugação enquanto a outra não, então um mesmo agente pode ser associado a dois pares diferentes, como se ilustra na figura-4.1-b.

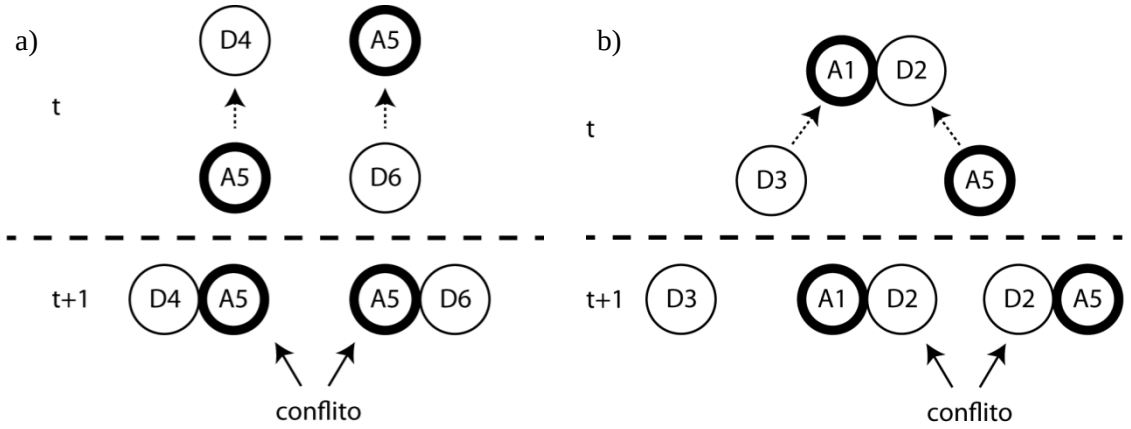


Fig. 4.1- Inconsistências típicas na configuração do sistema na iteração seguinte que podem emergir quando os *threads* aplicam as regras de interação independentemente. Se um determinado *thread* determinar o resultado da interação do agente D4, e um outro *thread* fizer o mesmo para o agente A5, então o primeiro *thread* vai determinar que o agente D4 formará um par com o agente A5, enquanto que o segundo *thread* vai determinar que o agente A5 vai formar um par com o agente D6. As configurações resultantes seriam inconsistentes. À direita apresenta-se uma inconsistência semelhante que ocorre quando *threads* distintos processam a dinâmica de interações de dois agentes num par ligado.

Estes conflitos nas decisões surgem porque as interações são efetuadas em paralelo e independentemente. Em princípio, a resolução destes conflitos poderia ser realizada permitindo que *threads* dispersos interagissem de forma a evitar a ocorrência dos conflitos mencionados. Este tipo de soluções requereria que *threads* diferentes acedessem à mesma região de memória o que levaria a atrasos de execução e pioraria o desempenho do algoritmo. De forma a evitar esta situação, o algoritmo deve ser construído de forma a que, o acesso a dados seja feita de forma tão independente quanto possível: diferentes *threads* devem usar diferentes entradas, para calcular diferentes saídas. A estratégia usada alcançar paralelismo no acesso a dados, consiste em impor que cada *thread* apenas realize uma decisão em cada iteração. O que é feito tendo em conta que, todos os agentes de um determinado tipo (por exemplo apresentadores), interajam com uma permutação aleatória de agentes do outro tipo. Deste modo podemos garantir que cada *thread* apenas determinará o resultado de uma decisão entre dois agentes. Como cada agente, interage apenas com um agente do outro tipo, então podemos garantir que o cálculo não requer acesso a variáveis de que outros *threads* necessitam, e também que o resultado da interação alterará a configuração final em variáveis que não serão alteradas pelos outros *threads*.

Deve-se notar que a estratégia proposta cria uma dinâmica que não é necessariamente equivalente à dinâmica que seria mais natural estabelecer com um algoritmo sequencial. Num algoritmo sequencial seria porventura mais natural adotar uma sequência de operações em que, um determinado agente poderia ser solicitado e interagir consecutivamente com agentes diferentes. A estratégia aqui adotada impõe que todos os agentes interajam simultaneamente, cada um com um agente diferente. De qualquer forma confirmei, usando um algoritmo sequencial, que ambas as estratégias levam a resultados qualitativamente equivalentes.

Um outro aspeto a salientar é que, de acordo com a presente estratégia, o estado de um agente pode ser alterado por dois *threads* diferentes em cada interação, tal como se exemplifica na Figura 4.2. Com efeito, apesar de só um *thread* determinar, através das regras de interação, a decisão desse agente, o seu estado depende também do resultado das interações nas quais o seu par participe. É o que sucede na Figura 4.2, com o agente A1 que acaba por ficar não ligado porque, o *thread* 3 determina que o seu par, o agente D4, se ligue ao agente A4.

Neste algoritmo têm prioridade a decisão que cada *thread* estabelece que origina novos pares. Estas decisões de alteração dos estados de agentes são marcadas a 1, conforme se representa na Figura 4.2. Isto determinará o estado dos agentes que se ligaram num novo par. Os restantes agentes serão mantidos no estado anterior, caso o agente a que estejam ligados tenha permanecido nesse estado ao fim do passo anterior.

A resolução dos conflitos nas decisões requer o uso de permutações aleatórias para estabelecer os pares de agentes que vão interagir. A criação de uma permutação pseudoaleatória em paralelo não é fácil. Na secção seguinte serão descritas as formas encontradas para a resolução deste problema.

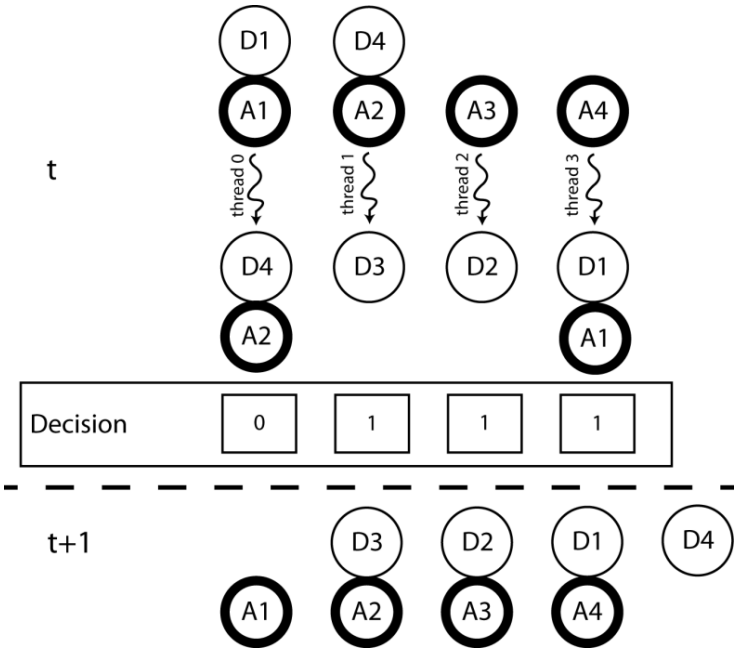


Fig. 4.2- De forma a evitar inconsistências e permitir uma execução independente entre os *threads*, os apresentadores (agentes do tipo A) interagem com uma permutação aleatória dos detetores sendo associados a diferentes *threads*. Por exemplo, o agente A1, conjugado com o agente D1, é colocado em interação com o agente D4, que está conjugado com o agente A2 e assim sucessivamente. Cada *thread* aplicará as regras da dinâmica para estabelecer a configuração da população na iteração seguinte. Se a decisão for a de estabelecer um par conjugado, então esta é aplicada imediatamente aos agentes envolvidos. Este é o caso dos agentes A2 e D3, A3 e D2 ou A4 e D1, na figura. Caso isto não aconteça (tal como na interação entre os agentes A1 e D4), então a decisão não terá qualquer impacto e consequentemente os pares envolvidos (D1A1 ou D4A2) apenas terminarão se as interações dos agentes com os quais os agentes A1 e D4 estão conjugados assim o determinarem. É o que sucede com estes agentes que acabam por ficar desligados.

4.2 Geração de permutações aleatórias no GPU

O *GPU* é uma unidade de processamento autónoma com a qual o *CPU* pode comunicar e trocar dados. Uma solução para gerar a permutação aleatória consistiria em gerá-la no *CPU*, e em seguida enviá-la para o *GPU*. Isto teria vantagens e desvantagens. Se por um lado a solução é de fácil implementação, por outro requereria enviar dados para o *GPU* a cada iteração. Ora as comunicações entre *CPU* e *GPU*, apesar de possíveis, devem ser minimizadas pois são lentas e levam a desempenhos inferiores. Por essa razão, optou-se por gerar a permutação no *GPU*. Existem na literatura, geradores de permutações aleatórias para cálculo paralelo. Verificámos no entanto, que estes geradores não geravam todas as permutações possíveis com *N* elementos. Isto

poderia ser problemático na presente aplicação pois implicaria que alguns agentes nunca interagissem entre si. Tornou-se por isso necessário desenvolver um novo algoritmo de criação de permutações aleatórias.

A ideia do algoritmo que desenvolvi consistiu em utilizar geradores de números aleatórios independentes em cada *thread.* Cada gerador, inicializado de forma arbitrária no início do algoritmo através de números aleatórios gerados no *CPU*, gera um número aleatório em cada *thread* (Figura 4.3, topo).

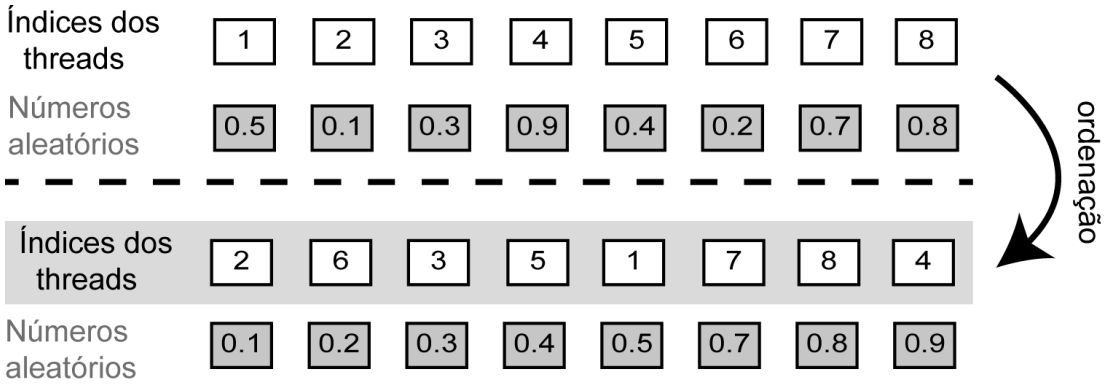


Fig. 4.3 – Ilustração do processo da geração de uma permutação aleatória.

Após esse processo, é executada a ordenação dos números aleatórios gerados usando uma estratégia também paralela. Cada *thread* deverá contabilizar quantos dos números aleatórios gerados pelos outros *threads* são inferiores ao número que ele gerou. Para evitar conflitos no acesso aos dados, cada *thread* acederá a uma posição diferente em cada iteração. O *thread i* começará por aceder ao número gerado pelo *thread i+1*, depois ao *i+2*, e assim sucessivamente. Quando aceder ao número gerado pelo último *thread*, começa a aceder aos números gerados pelos *threads* iniciais, conforme se ilustra na Figura 4.4. Este procedimento é geral, podendo-se aplicar a qualquer número de elementos que compõem a permutação. Este era aliás, um dos problemas dos geradores de permutações aleatórias existentes na literatura [16].

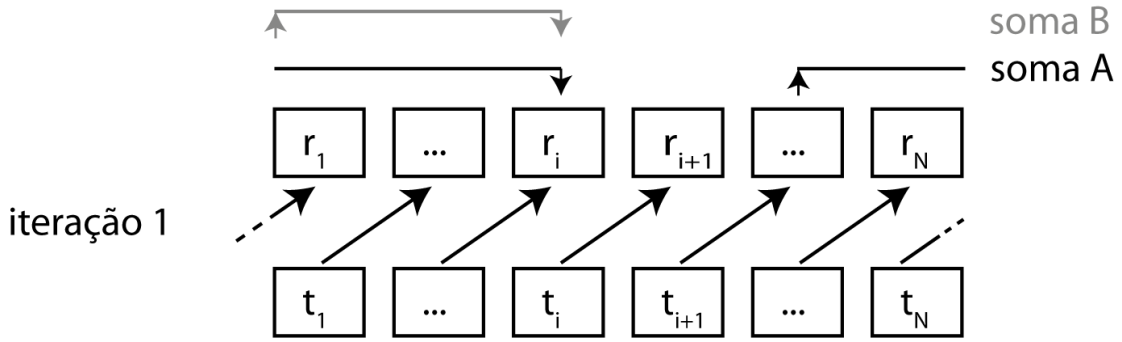


Fig. 4.4-Diagrama ilustrando o algoritmo de ordenação usado. O número de valores pseudo aleatórios inferiores a r_{i+1} (soma A), mais o número de valores pseudo aleatórios iguais a r_{i+1}, mas com um índice associado ao *thread* inferior a t_{i+1} (soma B), dá a posição para a qual se escreve o índice do *thread* t_{i+1}. De realçar que em cada iteração, *threads* diferentes acedem a posições diferentes.

4.3 Fluxograma do algoritmo de frustração celular

O algoritmo de frustração celular encontra-se representado na figura 4.5. Nesta figura, as tarefas foram separadas consoante são executadas no *CPU* (à esquerda) ou no *GPU* (à direita). O algoritmo de frustração celular começa por inicializar os agentes e copiá-los para o *GPU*. Após esse passo é lançada a função (*kernel*) de geração de matriz de interação.

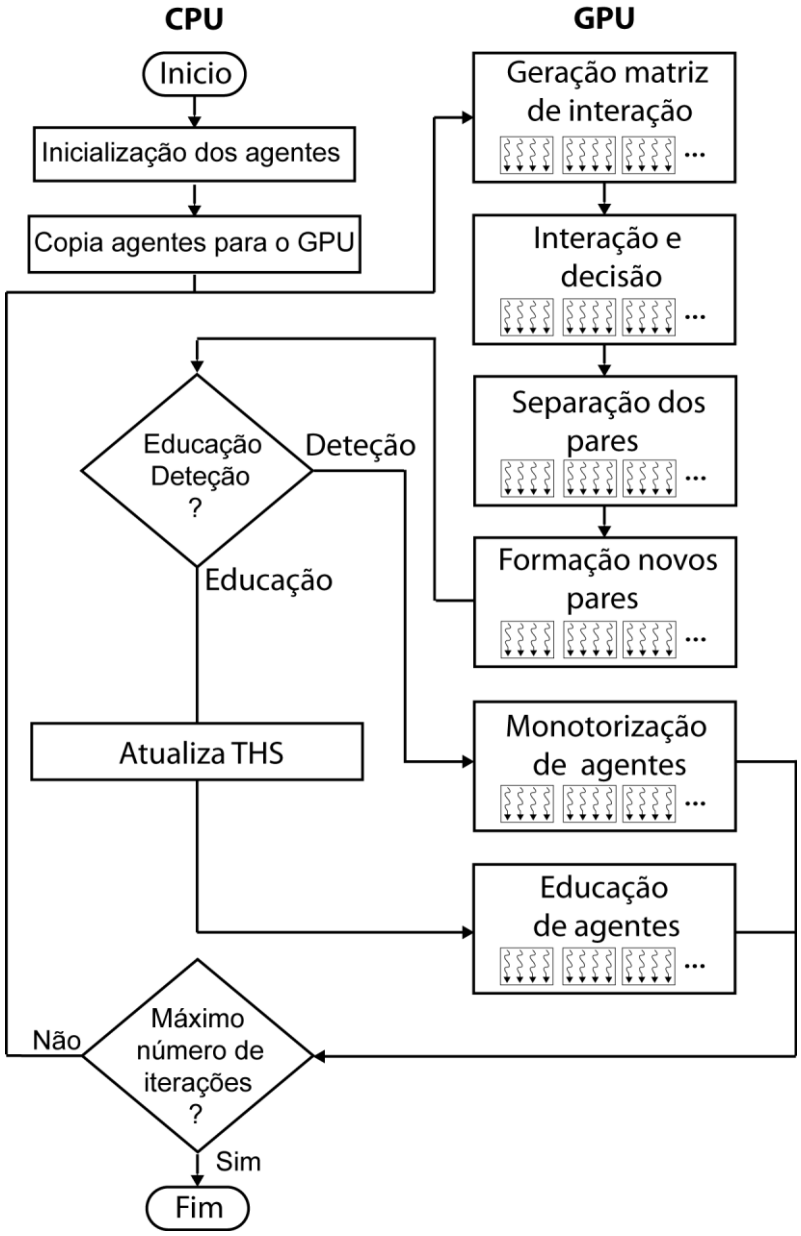


Fig. 4.5- Fluxograma do algoritmo de frustração celular paralelizado. À esquerda encontram-se as tarefas executadas no *CPU*, enquangto que à direita aparecem as tarefas executadas no *GPU*.

A matriz de interação consiste numa permutação aleatória dos índices dos agentes de um determinado tipo e estabelece que agentes dos diferentes tipos interagem.

A interação dos agentes consiste de várias etapas. Na primeira calculam-se as pontuações e decide-se, de acordo com as regras de interação (Definição ***3.3.3***) quais os agentes que querem formar um novo par. De seguida, os agentes que formem um novo par são separados dos seus antigos pares, são registadas as durações destes contactos e são formados os novos pares.

Após a interação dos agentes, o algoritmo aplica o processo de educação de repertório ou deteção (monitorização), dependendo do que se pretenda. A educação dos agentes detetores substitui agentes detetores envolvidos em interações pouco frustradas, por outros com recetores aleatórios. O objetivo é o de maximizar a frustração na população dos agentes em interação. Para tal, os pares que estejam ligados há mais do que um número de iterações designado de *THS (threshold),* são separados e o recetor do agente detetor substituído por outro aleatório.

O valor do *THS* é periodicamente atualizado de *w* em *w* iterações no *CPU*. Podem ser usadas várias estratégias para atualizar o seu valor. Uma das mais simples consiste em usar um valor fixo, e suficientemente pequeno. No entanto, isto apresenta a desvantagem de requerer o seu conhecimento prévio. Alternativamente, o *THS* pode começar num valor alto (por exemplo, igual à dimensão da janela *w*) e ir sendo atualizado sempre que os tempos de vida dos pares conjugados não excederem o seu valor num número de iterações *w*. Neste caso, o *THS* toma o valor do maior tempo de vida registado durante esse número de iterações.

A deteção é realizada monitorizando os tempos de vida das ligações dos agentes apresentadores durante um número de iterações *w*. Uma deteção de intrusão é realizada quando um agente apresentador realiza ligações longas com maior frequência do que sucede quando a população acaba de terminar o processo de educação do repertório. Para determinar o valor deste número típico, deve-se executar o programa de deteção sem intrusos (com uma população controlada) e determinar o número de ligações que um agente apresentador realizou com duração superior ou igual a um tempo pré-estabelecido (Definição ***3.3.7)***.

4.4 Performance computacional

Nesta secção abordaremos o desempenho em termos de velocidade de processamento do algoritmo paralelizado. Utilizamos um sistema de teste composto por um Intel core i7- 980 (*CPU*) a 3.33GHz, com 24GB de memória RAM (DDR3) a 2000MHz, e uma placa gráfica dedicada à computação. A placa gráfica usada foi a GTX-580 com 1.5GB de memória RAM (GDDR5) da NVIDIA. De forma a beneficiar de uma boa avaliação da performance, executou-se a mesma implementação em *GPU* e *CPU*. A implementação em *CPU* é uma versão sequencial do código traduzida do algoritmo paralelizado para o *GPU*. A tradução é feita recorrendo ao *Low Level Virtual Machine* (LLVM) providenciado pelo *GPU* Ocelot [17]. Relativamente ao *GPU*, foram considerados dois casos. No primeiro caso considerou-se apenas a dinâmica de uma população de agentes de cada vez, o que não utilizou todos os recursos do *GPU*. No segundo caso executou-se a dinâmica de 16 populações de agentes independentes em simultâneo. O número de populações escolhido foi determinado pelo número de processadores (16 *streaming multiprocessors*) na gráfica utilizada. De forma a comparar as performances computacionais, foi calculado o tempo requerido para executar a dinâmica de 16 populações durante 10000 iterações (tabela – 4.1). Foram ainda consideradas populações de diferentes tamanhos. Neste caso, e de forma a tirar partido do *GPU*, foram consideradas populações com um número de agentes múltiplo de 32 [13, 14]:128, 256, 512, 1024 e 2048. O tempo total requerido encontra-se apresentado na tabela 4.1 e na figura 4.6 mostra-se o ganho computacional do *GPU* relativamente ao *CPU*.

Tabela 4.1-Tempo necessário para simular a dinâmica de 16 populações durante 10000 iterações, para os diferentes dispositivos, e considerando um número de agentes total diferente.

Número de iterações	10000		
Número de populações a educar	16		
Dispositivo	*GPU*		*CPU*
Número de populações educadas em simultâneo	1	16	1
Número de agentes	**Tempo requerido em (s)**		
128	10.52	5.35	205.41
256	11.55	5.64	389.59
512	13.86	5.70	760.67
1024	22.10	6.64	1590.06
2048	41.66	8.68	3530.92

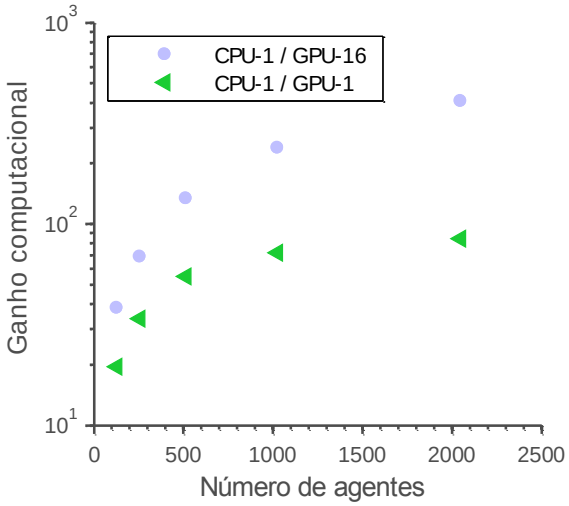


Fig. 4.6- Ganho computacional do processamento em *GPU* relativamente a *CPU* em função dos vários números de agentes. A razão CPU-1/GPU-16 mede o ganho computacional entre a situação em que a simulação da dinâmica de 16 populações é realizada uma de cada vez no *CPU* (CPU-1) ou é feita em simultâneo no *GPU* (GPU-16). A razão CPU-1/GPU-1 compara o processamento quando só uma população é simulada.

Conforme se constata na figura 4.6 e na tabela 4.1, a implementação em *GPU* reduziu significativamente o tempo computacional requerido. Os ganhos computacionais obtidos foram maiores para sistemas com um elevado número de agentes. Por exemplo, quando o número total de agentes é de 2048, o ganho computacional obtido foi de aproximadamente 406 vezes. Mas mesmo que se considere um sistema com um pequeno número de agentes (128), obtêm-se ganhos de aproximadamente 38 vezes. Estes ganhos são obtidos quando se recorre à versão do *GPU* que simula a dinâmica de 16 populações independentes em simultâneo. No entanto, os ganhos reduzem-se para apenas metade quando se simula a dinâmica de uma população de cada vez. Este aspeto não deixa de ser extraordinário. A característica mais notável do processamento

em *GPUs* consiste no pequeno aumento do esforço computacional requerido quando se aumenta o número de agentes. Por exemplo, o tempo requerido para a tarefa de simular as 16 populações com um número de agentes de 128 é de 5.35s, subindo somente para 8.68s quando se considera populações com um número de agentes de 2048. Isto é, ainda que o tamanho das populações tenha aumentado 16 vezes, o esforço computacional não aumentou sequer para o dobro. Isto contrasta claramente com o que se passa com o *CPU*, no qual o esforço computacional cresce mais rapidamente que o tamanho da população. Para o exemplo anterior, o aumento de 16 vezes no tamanho da população causa um aumento do esforço computacional de 17 vezes

4.5 Considerações finais

Considero que a paralelização do algoritmo de frustração celular foi uma das maiores contribuições deste trabalho. Como se verificou, para poder usufruir das potencialidades de cálculo disponibilizadas pelas placas gráficas, o programador deverá estar particularmente atento para incorporar na programação aspetos do *hardware*. Este facto torna a tarefa por vezes complicada e à data atual, específica da implementação considerada. Por isso, cada implementação pode tornar-se num desafio entusiasmante ainda que não acessível à generalidade dos programadores.

Como se viu neste capítulo, a paralelização pode requerer a transformação do modelo "físico" inicial numa versão computacional não exatamente idêntica. Ainda que a passagem entre o modelo físico e o computacional ocorra sempre que se implementam simulações computacionais usando métodos numéricos, os algoritmos paralelizados requerem uma nova geração de métodos e a criação de uma nova geração de modelos computacionais. Em todo o caso, o importante é que se confirme que o comportamento físico dos sistemas, idealizado e modelado, seja equivalente, de forma a que a distinção entre os dois se torne impercetível.

Ainda que os ganhos computacionais conseguidos tenham sido consideráveis, a estratégia que aqui se apresentou pode ser melhorada, nomeadamente no que respeita aos acessos à memória global. Também considero ser possível melhorar a criação de listas aleatórias. De facto, como apresentado cada *thread* tem de efetuar um número de passos igual ao número de agentes de cada tipo, para determinar a posição onde escreverá o número do índice. Acredito que isto é computacionalmente ineficiente e pode ser melhorado.

Outros desafios prender-se-ão com a implementação de modelos de frustração celular mais gerais e que podem levar ao desenvolvimento de estratégias de paralelização diferentes. Isso pode suceder, por exemplo, caso se pretenda implementar modelos em que cada agente interaja só com um conjunto limitado de agentes, em vez da situação descrita, em que a conectividade era total.

Capítulo 5: Discriminação *self-nonself*

A discriminação *self-nonself* é uma das tarefas mais complexas que o sistema imunitário tem de resolver. Não só deve atacar rapidamente uma enorme diversidade de patogénios, como evitar ao mesmo tempo, a autoimunidade, isto é respostas imunes contra o próprio corpo (*self*). Para evitar a autoimunidade, o sistema imunitário desenvolveu um período de treino no timo, para as células T que provêm da medula óssea. Durante este período mais de 95% das células que ali chegam são eliminadas por apoptose [6].

É também no sentido de evitar a autoimunidade que o modelo de frustração celular considera a existência de uma etapa de educação. Esta etapa de educação tem por objetivo aumentar a frustração das interações no sistema celular, minimizando os tempos de conjugação entre os agentes detetores e os apresentadores de *self*. Após esta etapa, os agentes detetores estariam preparados para discernir *self* de *nonself*. Durante bastante tempo verificamos que este mecanismo não era suficiente para obter uma discriminação quase perfeita pois a educação permitia que em muitas circunstâncias algumas células T estabelecessem ligações longas. Como resultado, as taxas de não deteção de agentes estranhos podiam chegar a 20%.

O mistério foi resolvido durante os trabalhos desta tese, com a introdução de um mecanismo designado por anergia e que ocorre também no sistema imunitário, ainda que não haja uma explicação consensual para a sua função [18]. O mecanismo da anergia aparece naturalmente no modelo de frustração celular. Consiste em considerar que existe um repertório extenso de células T e que só uma fração delas está presente nos nódulos linfáticos em cada momento. No decurso da dinâmica frustrada sempre que células T terminem ligações, são colocadas em anergia (i.e., inativadas) e substituídas por outras presentes no repertório. A lógica é simples: se existirem células T mal-educadas mesmo que em pequeno número, a probabilidade de serem substituídas por outras também mal-educadas é pequena, pelo que os resultados melhorarão.

Neste capítulo vamos confirmar que efetivamente a discriminação *self-nonself* pode ser perfeita no sistema imunitário. Este resultado terá também implicações no desenvolvimento de sistemas imunológicos artificiais. O que faremos é educar uma população de agentes apresentadores que apresentem um conjunto de sequências com *n* dígitos binários arbitrários. Posteriormente, analisaremos a dinâmica com a mesma população mas em que um agente apresentador possa apresentar uma sequência que não foi apresentada antes.

A natureza do resultado que aqui se apresenta requer que esta análise seja exaustiva. Isto é, para que possamos garantir que a taxa de não deteção é nula, consideraremos que a sequência apresentada pode ser qualquer uma das sequências não apresentada antes e disponível no espaço. Este estudo é computacionalmente intensivo mas não pode ser substituído por um estudo analítico. Com efeito, é diferente dizer que num espaço com 2^{20}~10^6 sequências estranhas nenhuma consegue evitar ser detetada, ou que em média isso não sucederá, como se faria numa análise de campo médio.

5.1 Educação

A educação do repertório reproduz o processo de seleção negativa que ocorre no sistema imunitário, no timo. No início do processo todos os agentes detetores possuem recetores aleatórios. Depois é iniciada a dinâmica do algoritmo, durante a qual, todos os pares que estejam formados há mais do que um número de iterações, *THS*, são separados e o recetor do agente detetor substituído por outro aleatório. O valor do *THS* é iniciado com um valor arbitrariamente

grande. O valor do *THS* é atualizado de *w* em *w* iterações, sempre que durante essas *w* iterações não tenha ocorrido nenhuma substituição. Este processo é repetido até que um número máximo de iterações ou um valor do *THS*, pré-estabelecidos, sejam atingidos. Na Figura 5.1a apresenta-se um gráfico com a evolução típica do valor do *THS* ao longo das iterações. Os resultados obtidos nesta tese usaram tipicamente 10^7 de iterações. No final deste processo guarda-se o repertório de agentes detetores obtido. Para criar um repertório de agentes detetores, este processo deve ser repetido várias vezes. Em cada uma, uma população de agentes detetores é guardada. Por simplicidade, mediremos a dimensão do repertório através do número de vezes que o processo foi repetido, ou seja, através do número de populações registado no repertório.
Durante a educação a duração dos contactos diminui. Este facto pode ser verificado na Figura 5.1b, que ilustra a evolução da média da probabilidade de um acontecimento com um tempo de vida superior a τ ocorrer ao longo de várias iterações (a laranja representam-se as primeiras iterações, em tons de azul as últimas). Na Figuras 5.1c e 5.1d ilustram-se também os histogramas cumulativos correspondentes para cada agente apresentador em duas iterações, uma, no início (à esquerda) e a outro na última iteração do processo (à direita). Estas curvas medem a probabilidade de um agente apresentador realizar uma ligação com uma duração superior a τ.

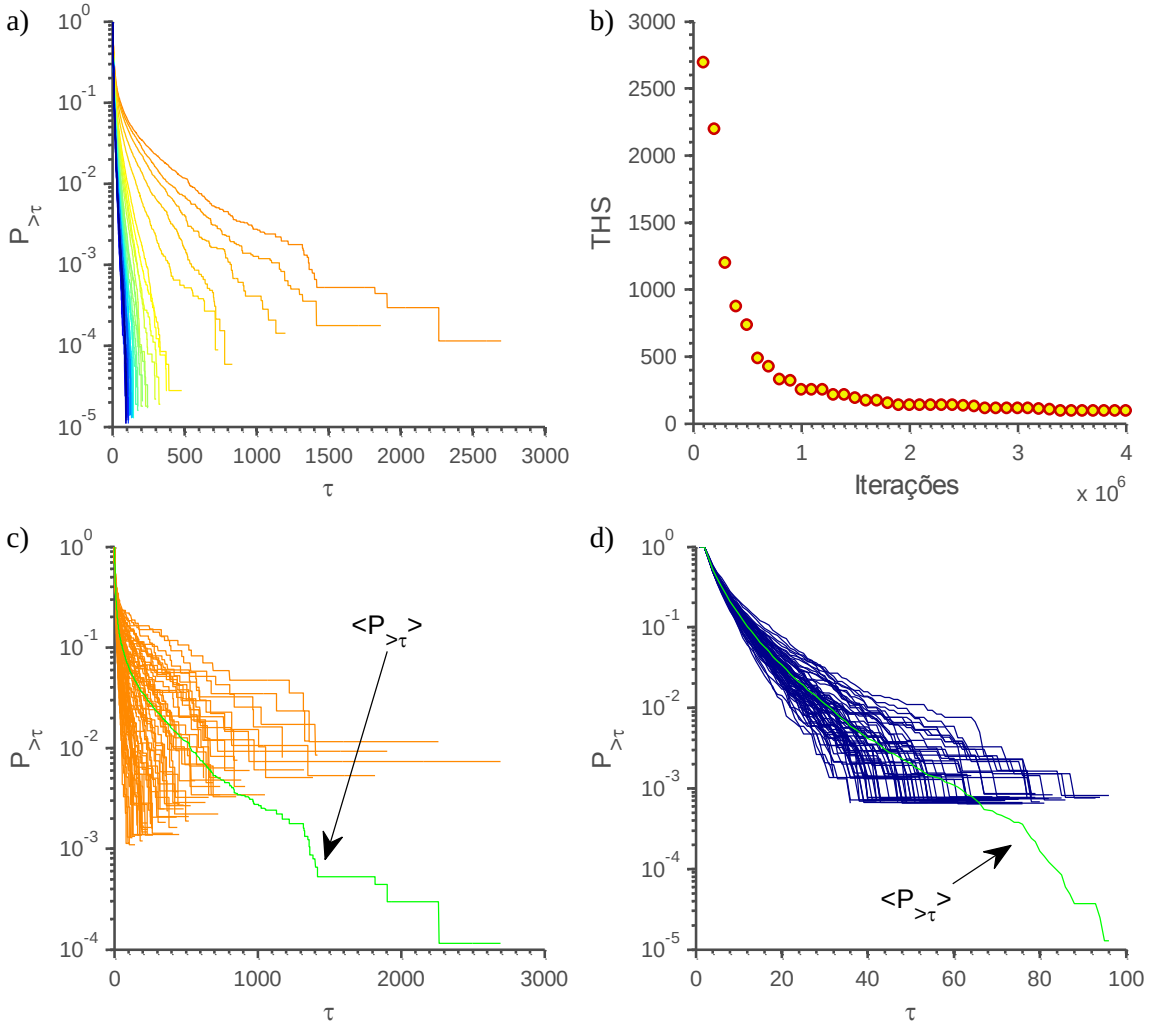


Fig. 5.1-Efeito da etapa de educação. Em a) a evolução da média da probabilidade de um acontecimento com um tempo de vida superior a τ, para um sistema com 128 agentes, uma janela de atualização de *THS* de $w=10^4$, durante 10^7 iterações. Em b) a evolução do *THS* durante a educação. Em c) e d) ilustra-se os histogramas correspondentes a cada agente apresentado, no inicio e no fim respetivamente.

Estes resultados são muito interessantes pois mostram que mediante o processo de seleção implementado, é possível transformar a dinâmica de interações da população. Considerando o tempo para o qual a curva média vale 10^{-3}, podemos verificar que a duração dos contactos diminuiu numa ordem de magnitude (de 1000 para 100 iterações). Isto mostra como o sistema é plástico, i.e., como a frustração dos agentes pode ser alterada mediante uma alteração da sua composição. Este resultado confirma efetivamente que faz sentido discutir o conceito que levou à formulação do princípio da máxima frustração [9]. Segundo este conceito, uma população organizada de forma a estabelecer contactos muito curtos, alterará fortemente a sua dinâmica caso se introduza no sistema um agente que não tinha sido educado de acordo com a dinâmica do sistema educado, maximamente frustrado. Na próxima seção confirmaremos que sistemas maximamente frustrados podem, efetivamente, permitir detetar sem erros a introdução de novos agentes.

5.2 Monitorização

Durante a fase de monitorização estudou-se a capacidade do algoritmo detetar sequências ausentes na fase de educação. Em particular, calculou-se quantas sequências estranhas conseguem não perturbar a dinâmica frustrada do sistema e escapar à deteção.

Para estabelecer um critério de deteção comparou-se a frequência com que os agentes apresentadores fazem ligações longas no fim da educação e na fase de monitorização. Designando por $P_{>\tau^*}^{Self}$ e $P_{>\tau^*}^{Mon}$ a probabilidade de um agente realizar ligações com duração superior a τ^*, durante a fase educação e monitorização, respetivamente, pode-se definir-se o seguinte rácio suscetível de utilização como critério de deteção:

$$D_{\tau^*} = \frac{\mathrm{Max}\{P_{>\tau^*}^{Mon}\}}{\mathrm{Max}\{P_{>\tau^*}^{Self}\}} \tag{5.1}$$

Caso D_{τ^*} seja superior a 1 então há ligações que ocorrem mais frequentemente na fase de monitorização, indicando que o sistema está menos frustrado e por isso sinalizam uma intrusão. Esta sinalização parte dos agentes para os quais $P_{>\tau^*} > \mathrm{Max}\{P_{>\tau^*}^{Self}\}$. Nos estudos que se seguem, e dado que conhecemos qual o agente que apresenta a sequência estranha, analisamos o rácio:

$$R_{\tau^*} = \frac{P_{>\tau^*}^{Intruso}}{\mathrm{Max}\{P_{>\tau^*}^{Self}\}} \tag{5.2}$$

onde $P_{>\tau^*}^{Intruso}$ representa a probabilidade do agente apresentador com a sequência estranha realizar ligações com duração superior a τ^*. O tempo de análise τ^* é escolhido como sendo suficientemente longo para que a análise seja clara, isto é, produza rácios claramente maiores que 1. Os resultados não dependem porém de uma escolha criteriosa do seu valor.

Na figura 5.2 - a) representa-se $R_{\tau^*=80}$ para os 448 agentes estranhos possíveis num espaço de sequências de 9 bits, com 64 sequências definindo o *self* e com uma população de 64 agentes apresentadores e 64 agentes detetores. Na Figura 5.2-b) apresenta-se o exemplo de uma boa deteção, com um rácio de duas ordens de magnitude, enquanto que na Figura 5.2-c) se ilustra o exemplo de uma não deteção. No início dos trabalhos desta tese, as taxas de deteção que se conseguiam obter eram da ordem dos 80%, nos melhores casos. Mas algumas pistas indicavam ser possível obter deteções perfeitas. Nomeadamente os estudos em sistemas ideais indicavam que em sistemas maximamente frustrados a deteção era perfeita[9].

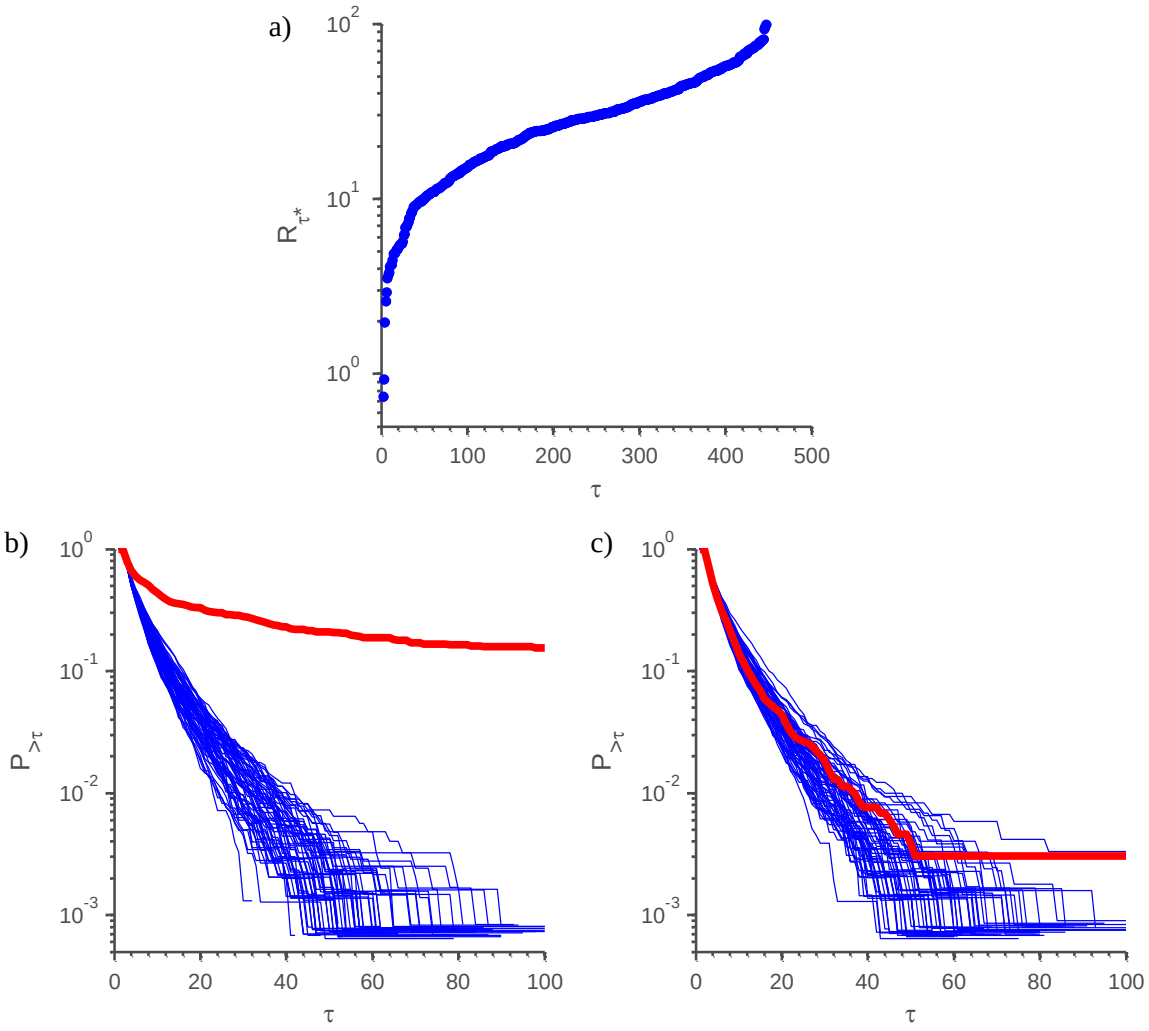


Fig. 5.2-Analise de deteção. Em a) a razão de deteção para todos os 448 agentes estranhos possíveis num sistema com 128 agentes, 64 apresentadores e 64 detetores, para um determinado tempo $\boldsymbol{\tau^* = 80}$. Em b) e c) o exemplo de uma boa deteção e de uma má deteção, respetivamente.

Durante o período desta tese surgiu a ideia de qual o efeito que seria necessário incorporar para atingir deteções perfeitas. Compreendeu-se não só que o mecanismo deveria permitir atingir melhores deteções, como também explicava o fenómeno da anergia em imunologia. A anergia no sistema imunitário verifica-se quando as células T que não recebem um sinal de co-estimulação, mesmo na presença de um patogénio, ficam sem reagir[18]. A justificação comum para este mecanismo é que as células T que saem do timo podem ainda exibir uma resposta imunitária contra antigénios do *self*, pelo que o mecanismo de anergia é uma forma de induzir tolerância. Da mesma forma, a etapa de educação presente no algoritmo de frustração celular não garante que os detetores tenham sido perfeitamente educados, pelo que pode ser útil um mecanismo adicional que induza tolerância sem reduzir a reatividade em relação a patogénios.

O mecanismo da anergia proposto substitui agentes detetores por outros no repertório e pertencentes ao mesmo *cluster*, sempre que os agentes detetores sejam abandonados numa ligação que tenha durado mais que um determinado tempo, designado tempo de anergia. Quanto menor este tempo, mais frequentes serão as trocas com outros agentes presentes no repertório.

Este mecanismo de anergia permite melhorar consideravelmente as taxas de deteção, pois substituem-se os agentes detetores que realizam os maiores tempos de ligação com alguns agentes apresentadores, por outros que, por uma questão de simetria na educação, terão pequena probabilidade de efetuar ligações longas exatamente com os mesmo agentes apresentadores. Se forem os agentes apresentadores a desencadear a resposta imunitária (ou no nosso caso, a acionarem o alarme de intrusões) então a probabilidade do mesmo agente apresentador produzir longos tempos de ligação é pequena e da ordem de $1/N_A$, onde N_A é o número de agentes apresentadores. Este efeito pode ser constatado na figura 5.3, onde é notória a redução da dispersão dos histogramas, e consequente redução da probabilidade de observar tempos longos. Ao contrário, no que respeita a um agente apresentando uma sequência estranha, a probabilidade do novo agente detetor fazer uma ligação longa é grande, e não dependerá da educação dos agentes detetores.

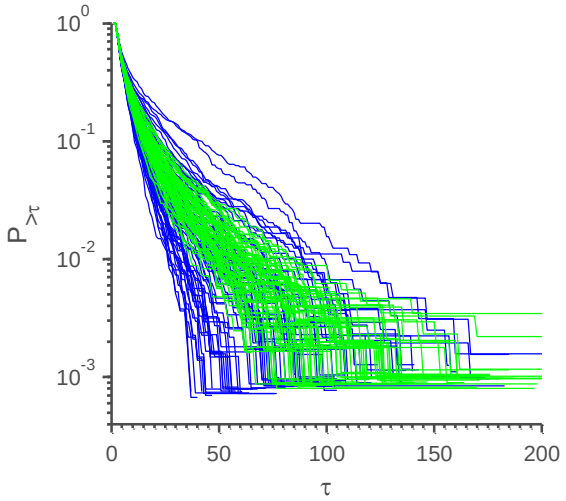


Fig. 5.3- Efeito da anergia (representado a verde) na probabilidade dos acontecimentos com um tempo de vida superior a τ. De notar que a azul se encontra a mesma probabilidade sem o efeito da anergia, e que o tempo de anergia foi de 5 iterações.

Após esta descoberta, um dos grandes objetivos desta tese consistiu em verificar se a taxa de não deteção poderia reduzir-se a zero, e em identificar os fatores que poderiam contribuir para que tal acontecesse. É o que se fará nas seções seguintes.

5.3 Resultados de discriminação *self-nonself*

5.3.1 Variação de espaço

Uma das grandes dificuldades verificadas nos métodos propostos na literatura [1], consiste na obtenção de discriminação *self-nonself*, principalmente para populações em espaços de dimensão elevada (i.e., para sequências com um grande número de bits). Nesses métodos, para se manter uma taxa de não deteção constante, o número de detetores cresce exponencialmente.

O propósito desta seção será o de mostrar que, no método de frustração celular, a taxa de não deteção pode ser nula, e que estes resultados são independentes da dimensão do espaço. Na Tabela 5.1 estão apresentados os resultados. Foram consideradas 64 sequências binárias aleatórias com *n* bits (*n*=7, …, 14) para cada apresentação por 64 agentes apresentadores. Efetuou-se a educação de 30 populações de agentes detetores durante $4x10^6$ iterações. A monitorização substituiu a sequência apresentada pelo primeiro agente apresentador por cada

uma das 2^n-64 sequências estranhas possíveis. O tempo de corte usado na monitorização, τ^*, foi escolhido tendo em conta o resultado do tempo de vida máximo observado no fim da etapa de educação.

Tabela 5.1-Resultados da discriminação *self-nonself*.

Número de *clusters*	8							
Janela de deteção *w*	10000							
Número de populações de agentes apresentadores	1							
Número de populações de agentes detetores	30							
Número de agentes por população	64							
Tempo de anergia	5							
$\boldsymbol{\tau^*}$	80							
Número de bits	7	8	9	10	11	12	13	14
Regra	**Taxa de não deteção (%)**							
TWR	0	0	0.22	0.31	1.11	0.87	1.14	0.97
#rcb	0	0	0	0	0	0.03	0.06	0
G^0G^1	0	0	0	0	0.05	0.05	0.05	0.02
Y	0	0	0	0	0	0.03	0.03	0.05

Todas as regras produziram resultados que superam largamente o que está publicado na literatura [1, 3] onde tipicamente as taxas de não deteção se situam entre 5% a 10%. A regra que obteve piores resultados para a taxa de não deteção foi a *TWR*. Como se verá à frente, estas taxas poderão ser ainda reduzidas, permitindo fazer a afirmação de que a discriminação perfeita é atingível.

Os resultados apresentados na Tabela 5.1 são típicos, no sentido em que, repetindo o processo de monitorização, eles são reprodutíveis. Para ilustrar este facto, foram educadas 30 populações de agentes detetores durante $4x10^6$ iterações, tendo-se repetido a monitorização 10 vezes. Em cada um dos casos registaram-se e ordenaram-se os rácios R_{τ^*} para cada uma das 16320 sequências estranhas possíveis. Para cada ponto destas curvas calculou-se o valor médio e o intervalo de confiança somando e subtraindo ao valor médio o erro padrão. O resultado está apresentado na figura 5.4.

Conforme se pode confirmar, o conjunto de sequências que pode escapar à deteção representa uma pequena fração dos rácios obtidos. A maior parte dos rácios são claramente superiores a 10.I.e., o número de ligações com a duração τ^* é pelo menos 10 vezes mais frequente no caso de agentes apresentando a sequência estranha, do que apresentando uma outra sequência. Estes resultados são, além disso pouco variáveis, conforme se pode verificar pela largura do intervalo de confiança. Finalmente, as várias regras podem produzir rácios ligeiramente diferentes, mas os resultados são qualitativamente idênticos.

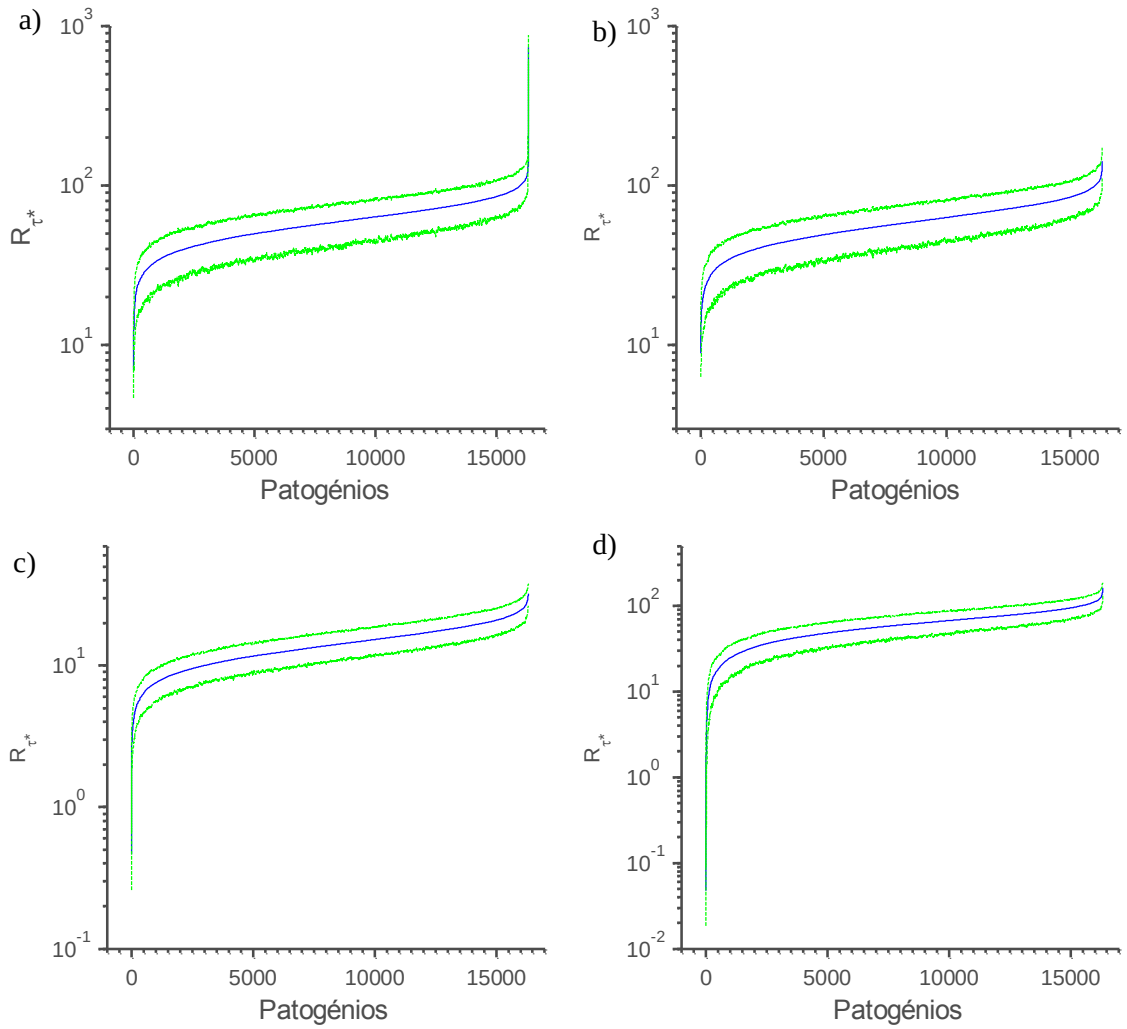


Fig. 5.4-Análise de deteção. Em a-b-c-d encontra-se representado a média (a azul) e o desvio padrão dos rácios de deteção para os 16320 agentes estranhos, para as regras *Y*, G^0G^1, *#rcb* e *TWR*, respetivamente.

O valor do rácio obtido, não é, na realidade muito informativo. Na realidade, ele pode ser alterado significativamente alterando o valor de τ^*. Isto porque, o que a capacidade de deteção estabelece é que o decaimento das curvas relativas à duração dos contactos envolvendo o agente com a sequência estranha e os demais, tem tempos de vida associados diferentes. Como resultado, a razão entre as duas curvas será também uma exponencial, divergindo para tempos longos. O importante será portanto que as curvas da Figura 5.4 tenham um patamar estável tal que o intervalo de confiança seja claramente superior a 1. De qualquer forma, para obter os resultados claros dessa figura foi necessário considerar um tempo τ^* suficientemente longo para que os rácios sejam grandes, mas curto para que haja ocorrências estatisticamente relevantes de todos os agentes no sistema, como se ilustra na figura 5.5 .

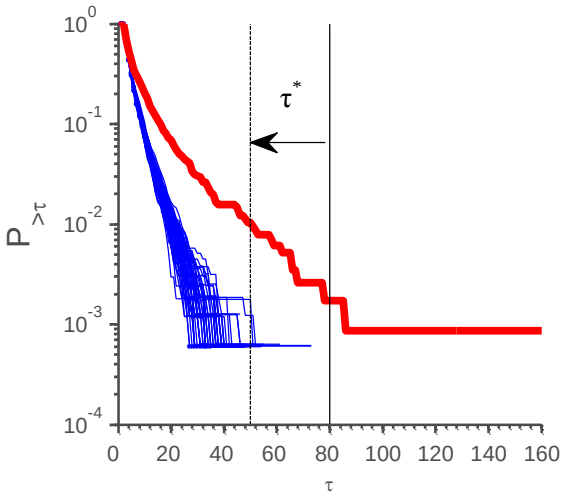


Fig. 5.5- Probabilidade de um acontecimento com um tempo de vida superior a τ, para todos os agentes do sistema (a azul), e para o patogénio (a vermelho).

Outra questão que se pode colocar relaciona-se com a variabilidade destes resultados com a população inicialmente apresentada e que define o *self*. Também aqui, os resultados são robustos. Na tabela 5.2 apresentam-se resultados para 10 simulações com uma população de sequências apresentada diferente. O conjunto dos resultados mostra uma considerável independência em relação à amostra usada nas sequências apresentadas.

Tabela 5.2-Taxa de não deteção para as diversas regras, considerando conjuntos de *self* diferentes.

Regra	*TWR*		*#rcb*		Y^0Y^1		*Y*	
Número de bits	7	14	7	14	7	14	7	14
Conjunto de *self*	**Taxa de não deteção**							
1	0	0.55	0	0	0	0.02	0	0.05
2	0	0.67	0	0	0	0.06	0	0.04
3	0	0.30	0	0	0	0.02	0	0.02
4	1.56	0.75	0	0.03	0	0.04	0	0.03
5	1.56	0.61	0	0.02	0	0.02	0	0.04
6	0	0.35	0	0.01	0	0.02	0	0
7	1.56	0.33	0	0.01	0	0.01	0	0.01
8	0	0.91	0	0.02	0	0.04	0	0.04
9	0	0.63	0	0.02	0	0.02	0	0.05
10	0	0.85	1.56	0.01	0	0.03	0	0.05

5.3.2 Efeito da educação

O método da frustração celular assenta num novo principio de organização celular, o principio da máxima frustração proposto em [9]. Uma questão importante, tanto de um ponto de vista conceptual como prático, será o de saber se a capacidade de deteção requer uma maximização da frustração muito exigente – vide, a determinação de repertórios perfeitamente frustrados - ou se os resultados se obtêm com relativa facilidade mesmo com processos de educação pouco elaborados. De salientar, que o mecanismo de seleção aqui proposto substitui um agente detetor por outro com recetor aleatório, ou seja, é um processo que não requer qualquer informação da parte do utilizador (ou do sistema imunitário).

Na Tabela 5.3, mostra-se que a maximização da frustração é importante para obter boas taxas de deteção. Para estes resultados considerou-se uma população apresentando 64 sequências binárias aleatórias com 14 bits, tendo-se educado o repertório dos agentes detetores,

com diferentes valores alvo do tempo máximo de ligação, *THS*. Posteriormente foi medida a taxa de não deteção para as várias regras de codificação das listas de interação.

Tabela 5.3-Efeito da qualidade de educação na taxa de deteção.

Número de *clusters*		8				
Janela de deteção *w*		10000				
Número de populações de agentes apresentadores		1				
Número de agentes por população		64				
Tempo de anergia		5				
τ^*		80				
Regra	**Nº de populações de agentes detetores**	**THS usado para educar**	400	300	200	100
TWR		**Nº iterações médio**	260000	270000	440000	900000
	30	**Taxa de não**	13.36	5.65	0.94	0.33
	60	**deteção (%)**	7.15	2.30	0.56	0.05
#rcb		**Nº iterações médio**	687500	895000	1567500	10000000 (máximo)
	30	**Taxa de não**	0.06	0.12	0.01	0.03
	60	**deteção (%)**	0.02	0.05	0.01	0.03
G^0G^1		**Nº iterações médio**	237500	300000	410000	912500
	30	**Taxa de não**	13.79	2.60	0.19	0.03
	60	**deteção (%)**	7.11	2.35	0.02	0
Y		**Nº iterações médio**	227500	292500	422500	895000
	30	**Taxa de não**	7.24	2.90	0.32	0.03
	60	**deteção (%)**	5.96	0.94	0.03	0.01

Os resultados mostram que se obtêm melhores taxas de deteção para sistemas melhor educados. Porém, a taxa de não deteção decai muito rapidamente. Na realidade se representarmos a taxa de não deteção em função do THS, o seu decaimento, é, na maioria dos casos, praticamente exponencial. Isto é interessante pois mostra que se não forem pretendidas taxas de não deteção exatamente iguais a zero, então o processo de educação não precisa de ser muito custoso.

5.3.3 Efeito do número de *clusters*

Uma das questões a que nos colocamos foi a de saber como se devia organizar a população de agentes de forma a maximizar a frustração e a taxa de deteção. Estudamos várias organizações, e por fim, convergimos na organização em *clusters*, que nos pareceu ser aquela que melhor correspondia ao que se conhece acontecer em imunologia. A questão computacional que se colocou foi então a de compreender se o número de *clusters* em que está organizado o sistema, pode influenciar os resultados.

Foram considerados populações com um número de *clusters* igual a 2, 4, 8 e 16. Para cada caso foi constituído um repertório com 30 populações de agentes detetores, sendo o *self*

constituído por 64 sequências binárias de 14 bits. Em seguida procedeu-se à monitorização dos 2^{14}-64 patogénios para cada população apresentando-se os resultados na tabela 5.4.

Tabela 5.4-Efeito do número de *clusters* na taxa de deteção.

Número de bits	14			
Regra	*TWR*	*#rcb*	G^0G^1	*Y*
Número *clusters*	**Taxa de não deteção (%)**			
2	16.11	98.57	0.30	0.20
4	0.06	0.03	0	0
8	0.33	0	0.02	0.05
16	4.87	0.12	1.34	1.93

Nota-se que a utilização de 2 *clusters* pode ser problemática, principalmente para as regras *TWR* e *#rcb.* Isto pode dever-se ao facto do sistema criar emparelhamentos estáveis com estas regras. O aumento do número de *clusters* tende a frustrar mais o sistema pois há mais interações que destabilizam, também pelo lado dos agentes apresentadores, as interações. Tendo em vista aplicações práticas, estes resultados indicam que o número de *clusters* a escolher deve ser maior que 2, e não há vantagem em ser grande.

5.3.4 Efeito da anergia

Sempre que um agente detetor realizar um tempo longo, podem suceder duas coisas: ou está a estabelecer uma ligação longa porque está a detetar um agente estranho, ou está mal-educado, porventura num certo contexto dos agentes que estão em interação. Nesse caso, a anergia permite substituir esse agente por outro, de forma a diminuir o impacto dessa má educação. Ao contrário, se o agente apresentador estiver a apresentar uma sequência estranha, então um número macroscópico de agentes detetores no repertório verá esse agente como estranho, e por isso uma ligação longa a esse agente apresentador se repetirá.
Uma questão que se pode colocar é se esta descrição do efeito da anergia beneficia efetivamente a tarefa de deteção? A outra prende-se com a questão de saber se a dimensão do repertório de detetores é crucial.

Para responder à primeira questão, estudou-se como a variação do tempo da anergia afeta a taxa de deteção. Para tempos de anergia grandes, só agentes detetores envolvidos em ligações muito longas são substituídos. Uma diminuição desse valor aumenta a taxa de renovação dos agentes detetores na população, e por isso diminui o impacto da presença de agentes detetores mal-educados. Na Tabela 5.5 mostra-se o efeito da variação do tempo da anergia em populações com um repertório de 30 populações de agentes detetores educadas durante $4x10^6$ iterações. O repertório do *self* usado foi o mesmo que o da secção anterior. Foram considerados diversos tempos de anergia 2, 5 10, 15 iterações. O efeito é notório e qualitativamente partilhado pelas várias regras. Os resultados são claros e mostram que tempos de anergia mais curtos beneficiam o desempenho da deteção de intrusões.

Tabela 5.5- Efeito do tempo de anergia na taxa de deteção para as diferentes regras.

Número de bits	14			
Regra	*TWR*	*#rcb*	G^0G^1	*Y*
Tempo de anergia	**Taxa de não deteção (%)**			
2	0.41	0.02	0	0.01
5	0.33	0	0.02	0.05
10	1.27	0.01	0.21	0.25
15	4.48	0.03	2.23	2.62
30	47.77	0.18	34.80	35.45
50	58.12	3.08	50.51	51.04
100	60.41	11.74	51.07	53.07

Finalmente, estudou-se o efeito da dimensão do repertório de agentes apresentadores. Neste caso repetiu-se a etapa de monitorização considerando repertórios com 1, 2, 3, 6, 10, 15, 30, 45, 60 populações de agentes detetores. Estes repertórios foram formados exigindo que o valor de *THS* fosse inferior a 100 para terminar a educação. Os resultados relativos à taxa de não deteção em função do número de populações de agentes detetores podem ser apreciados na tabela 5.6. Para todas as regras, verifica-se um decaimento da taxa de não deteção que se torna virtualmente nula para repertórios com 60 ou mais populações. Neste caso há a salientar que a regra *#rcb* é aquela que requer comparativamente repertórios mais pequenos. Este facto pode ter relevância prática.

Tabela 5.6- Efeito da variação do número de populações de agentes detetores usadas para efetuar a deteção de 16320 agentes estranhos, para as diversas regras.

Número de populações de agentes detetores	1	2	3	6	10	15	30	45	60
Regra	**Taxa de não deteção (%)**								
TWR	62.59	40.31	28.27	7.97	3.10	1.24	0.23	0.13	0.07
#rcb	18.38	3.86	1.81	0.15	0.04	0.02	0.01	0	0
G^0G^1	50.47	30.08	20.81	7.99	2.27	0.50	0.04	0.01	0
Y	46.77	24.25	13.99	4.52	0.99	0.23	0.02	0	0

5.3.5 Variação do tamanho do *self*

Os dados apresentados nas secções anteriores, levavam em consideração que o número de agentes por população era igual ao número de sequências que compunham o *self*. No entanto, pode ser necessário considerar sistemas em que o *self* seja constituído por um número de elementos maior. Nesta secção são apresentados dois métodos para o fazer.

O método mais simples para aumentar o *self* consiste em aumentar o número de agentes por população. No entanto, como se verá, este método tem limitações. Para o compreender foram consideradas populações com um número de agentes por população de 32, 64,128 e 256. Em cada caso efetuou-se a educação de 30 populações de agentes detetores durante $10x10^6$ iterações. O *self* considerado na etapa de educação foi constituído por 32, 64, 128 e 256 sequências binárias de 14 bits (ver tabela 5.7).

Tabela 5.7- Taxa de não deteção em função do número de agentes usado por população.

Regra	*TWR*	*#rcb*	G^0G^1	*Y*
Número de agentes por população	**Taxa de não deteção**			
32	1.11	0.05	0.24	0.24
64	0.23	0.09	0.01	0.02
128	0.57	0.02	0.01	0.01
256	2.36	2.28	0.10	0.11

O resultado mais assinalável é que mesmo para 256 sequências diferentes, as taxas de não deteção são ainda bastante baixas e muito inferiores às apresentadas na literatura. Essa conclusão é particularmente válida para as regras baseadas em geradores de números aleatórios. A maior complexidade destas regras parece ser importante para conciliar a tolerância em relação ao elevado número de sequências do *self*, e simultaneamente, a alta reatividade em relação às sequências estranhas.

Um outro aspeto a assinalar está no facto dos resultados obtidos para 32 agentes não serem os ótimos. Isto deve-se ao facto de se reduzir a diversidade de agentes detetores no repertório, uma vez que este é também mais reduzido.

Finalmente, um outro aspeto a referir relaciona-se com o facto de que um aumento do número de agentes que forma o *self* tende a requerer etapas de educação mais demoradas, ou seja com mais iterações e com janelas de educação, *w*, maiores. Isto resulta de que, aumentando o número de sequências presentes na população, será necessário estabelecer uma ordenação de listas mais apurada, o que é difícil através da introdução constante de listas aleatórias.

Um outro método que permite aumentar a quantidade de sequências apresentadas no *self* consiste em trocar as sequências apresentadas. As sequências são trocadas, sempre que os agentes apresentadores sejam abandonados após terminarem ligações com um tempo de vida superior a um valor de corte. Este mecanismo foi inspirado no mecanismo da anergia, embora se aplique aos agentes apresentadores. Este tempo de corte é denominado por tempo de substituição do agente *Apresentador*.

Os resultados que se apresentam na Tabela 5.8 foram obtidos com populações em que cada agente apresentador pode apresentar até 10 sequências de 14 bits. Usaram-se repertórios com 30 populações de agentes detetores educadas durante $10x10^6$ iterações. O tempo de substituição do agente *Apresentador* considerado foi de 5 iterações.

Tabela 5.8- Taxa de não deteção em função do número de populações de agentes apresentadores.

Número de populações de *self*	1	2	3	4	5	6	7	8	9	10
Tempo de substituição do agente Apresentador	5									
Regra	**Taxa de não deteção (%)**									
TWR	0.10	0.21	0.71	3.86	8.05	16.79	24.61	28.41	40.30	49.56
#rcb	0.01	0.07	0.07	0.99	1.12	2.01	2.34	4.72	6.33	13.95
G^0G^1	0	0	0	0.03	0.27	2.04	4.96	13.80	24.46	31.92
Y	0	0	0	0.02	0.20	2.12	5.82	14.50	25.01	35.84

Para um número de sequências apresentadas por agente apresentador inferior a 4 ou 5, as regras aleatórias obtém taxas de não deteção semelhantes às registadas anteriormente (tabela 5.1). Isto mostra a concordância nos resultados, pois o número de sequências apresentadas

nesses casos é de $4 \times 64 = 256$, ou seja, o número de sequências para as quais começa a taxa de não deteção a ser não desprezável no caso anterior. Para números maiores, as taxas de não deteção crescem, atingindo valores consideráveis (>30%).

Estes resultados mostram que, para que o algoritmo da frustração celular seja escalável, torna-se necessário adotar outras estratégias. Essas estratégias passam pela limitação da conectividade dos agentes, um tópico que está fora do âmbito do presente trabalho, pois obriga a alterações à estratégia de paralelização. No entanto, a demonstração de que dessa forma se obtêm os resultados desejáveis foi entretanto demonstrado pela minha colega Patrícia Mostardinha, usando algoritmos mais simples. Ou seja, podemos efetivamente garantir que o método pode ser aplicado a qualquer número de sequências que compõe o *self*.

Um outro aspeto a referir sobre a importância destes resultados foi a de que o algoritmo demonstrou ter uma capacidade de generalização diferente da que é conhecida noutros métodos, por exemplo em redes neuronais. Efetivamente, o método demonstrou ter tolerância total ao *self* o qual era composto por um número extraordinário de configurações distintas. Por exemplo, com 4 sequências apresentadas por agente apresentador, o número de configurações apresentadas foi de, potencialmente, $4^{64} \sim 10^{38}$. Esta capacidade de ganhar tolerância a tantas configurações distinta é notável.

Na realidade, o que ainda é mais notável é que o sistema consegue distinguir configurações diferentes. Na figura 5.6 ilustra-se este facto. Educou-se uma população com 64 agentes que apresentavam ou sequências de um conjunto, A, ou de um outro, B, mas de tal forma que de *w* em *w* iterações todas as sequências apresentadas eram alteradas de uma população para a outra. Durante a fase de monitorização apresentou-se uma população em que metade das sequências pertencia ao conjunto A e a outra metade ao conjunto B. Os histogramas obtidos e apresentados na figura 5.6 mostram que a dinâmica é profundamente alterada na segunda configuração.

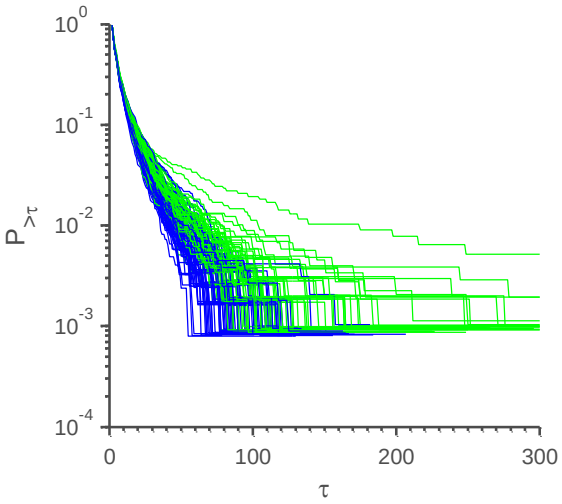


Fig. 5.6-A azul a dinâmica do sistema educado com duas configurações, A e B, e a verde a dinâmica de uma configuração constituída por metade de A e metade de B.

Estes resultados demonstram que a definição de *self* e *nonself* tem de ser alargada. O sistema não só aprende que sequências pertencem ou não ao sistema, como também aprende que configurações são as normais. Segundo a definição de deteção através do rácio D_{τ^*}, o sistema terá acionado o alarme de intrusões na configuração mista, apesar de todas as sequências apresentadas serem consideradas, individualmente, *self*. Na realidade, o sistema acusa o aparecimento de uma configuração *self* anormal.

Este tipo de sinalização é impossível de atingir com os sistemas propostos na literatura [1] e apresentados no capítulo 2. Nestes modelos a deteção de intrusões envolve somente o reconhecimento de uma sequência por um agente e é independente da configuração da população. Porém, a necessidade de definir o *self* como uma propriedade do sistema parece ser mais natural. Isso é aliás evidente quando se pensa na deteção de anomalias num texto escrito. Se pensarmos nas palavras como as sequências a apresentar, o aparecimento de erros ortográficos estaria ligado à deteção *nonself*, enquanto o aparecimento de erros gramaticais se relaciona com a deteção de uma apresentação anormal de *self*. Esta é uma área em que o nosso grupo está neste momento a desenvolver a maior parte da sua investigação, porque é também a menos trivial.

5.4 Considerações finais

Neste capítulo foi avaliado o desempenho do algoritmo de frustração celular com vista a aplicações na deteção de intrusões. O algoritmo é composto de duas etapas principais: educação e monitorização. A etapa de educação consiste em substituir agentes detetores por outros com recetores aleatórios. Trata-se de um processo não dirigido que não faz assunções sobre que sequências são *self* ou *nonself*. Este processo assume que a frustração do sistema de agentes em interação pode ser aumentada. Neste capítulo verificamos que assim se passa: os sistemas frustrados possuem uma enorme plasticidade permitindo, por seleção dos agentes presentes na população, modificar a frustração da dinâmica do sistema. O algoritmo da frustração celular usa a fase de educação para criar um repertório de detetores composto por várias populações de agentes que maximizaram a frustração do sistema.

A etapa de monitorização, avalia a dinâmica frustrada através da análise da duração dos contactos. A frequência de contactos longos assinala a presença de intrusões ou anomalias no sistema. Verificou-se que é necessário um mecanismo de anergia para atingir discriminação perfeita entre *self* e *nonself*. O mecanismo de anergia consiste em substituir agentes detetores que tenham terminado uma ligação e ficado não ligados, por outros presentes no repertório constituído durante a educação.

No capítulo 3 verificou-se que estudos analíticos serão sempre limitados na avaliação da qualidade com que o algoritmo de frustração celular desempenha a tarefa de discriminação *self-nonself*. Por isso este capítulo dedicou-se a estabelecer quantitativamente, e através de estudos computacionais, a qualidade da discriminação *self-nonself* atingida pelos sistemas celulares frustrados. Os algoritmos de frustração celular conseguem facilmente atingir taxas de não deteção inferiores a 0.5%, podendo ainda ser inferiores, e possivelmente perfeita, caso utilizem um repertório de detetores suficientemente grande e a educação do repertório tenha sido razoavelmente exigente.

Os estudos neste capítulo permitiram concluir ainda que, o efeito da regra, de definição implícita, das listas de interação não é determinante no desempenho, pois as várias regras conseguiram atingir resultados equivalentes. Isso mostrou a robustez do algoritmo, que não é sensível a muitos detalhes e não requer um ajuste fino de parâmetros.

Finalmente, também se mostrou que o algoritmo de frustração celular é sensível ao contexto e que a definição de *self* tem de passar a ser vista de uma forma mais abrangente. Assim, o *self* deverá ser definido como o conjunto de todas as sequências que o sistema apresenta e ainda a forma como são apresentadas. Estudos mais apurados nesta perspetiva farão parte de trabalhos futuros.

Capítulo 6: Conclusões

A questão de saber o sistema imunitário consegue discriminar *self de nonself* é uma questão que persiste em imunologia. Aliás os trabalhos realizados na área da inteligência artificial parecem indicar que esta tarefa dificilmente seria perfeita e por isso a sua utilidade para detetar intrusões seria questionável.

Neste trabalho começámos por introduzir a visão imunológica de como o sistema imunitário efetua esta discriminação. Posteriormente, mostrámos como foi feita a descrição computacional do mesmo mecanismo. No entanto, mostrámos que os modelos atuais não conseguem discriminar perfeitamente o *self* do *nonself*, pois as concetualizações atuais levam ao aparecimento de buracos no espaço das sequências apresentadas. Tipicamente as taxas de não deteção cifram-se em não menos que 5%, havendo ainda limitação no tamanho das sequências apresentadas.

Neste trabalho, estudou-se o algoritmo da frustração celular, proposto pelo orientador desta tese como alternativa aos algoritmos de seleção negativa existente. Foi analisada a sua relevância imunológica e desenvolvido um modelo computacional baseado nesta visão.

Na sua formulação original, o modelo da frustração celular usa a conceito de listas de interação. Nesta tese estudaram-se formas implícitas de definir estas listas de interação de forma a tornar o algoritmo mais rápido e menos oneroso em termos de espaço de memória. Em particular foram definidas pela primeira vez duas regras que usam geradores aleatórios.

O trabalho fundamental desta tese e totalmente original, prendeu-se com a paralelização do algoritmo da frustração celular e consequente utilização. A paralelização do algoritmo requereu transformar o modelo da frustração celular numa versão paralelizada - um novo modelo computacional – que conservasse o mesmo comportamento "físico".

Os ganhos computacionais atingidos foram consideráveis. Experiências que tipicamente demoravam um dia de computação, passaram a ser realizadas em poucos minutos. Como resultado, nesta tese foram apresentados resultados exaustivos sobre o comportamento dos algoritmos de frustração celular na deteção de intrusões. Nesta tese demonstrou-se que os algoritmos de frustração celular permitem atingir uma discriminação perfeita entre *self* e *nonself*, em condições relevantes para aplicações práticas.

Bibliografia

[1] S. Forrest, *et al.*, "Self-Nonself Discrimination in a Computer," *1994 Ieee Computer Society Symposium on Research in Security and Privacy, Proceedings*, pp. 202-212, 1994.

[2] Z. X. Xie, *et al.*, "A distributed agent-based approach to intrusion detection using the lightweight PCC anomaly detection classifier," *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, Vol 1, Proceedings*, pp. 446-453, 2006.

[3] P. Dhaeseleer, *et al.*, "An immunological approach to change detection: Algorithms, analysis and implications," *1996 Ieee Symposium on Security and Privacy, Proceedings*, pp. 110-119, 1996.

[4] D. Dasgupta and L. F. Niño, "Immunological computation : theory and applications." Boca Raton ; London: CRC, 2009.

[5] A. Hone, *et al.*, "Theoretical advances in artificial immune systems," *Theoretical Computer Science*, vol. 403, pp. 11-32, Aug 20 2008.

[6] A. K. Abbas and A. H. Lichtman, "Basic immunology : functions and disorders of the immune system", 2nd ed. Philadelphia: Saunders, 2004.

[7] D. Dasgupta, *et al.*, "Artificial immune system (AIS) research in the last five years," *Cec: 2003 Congress on Evolutionary Computation, Vols 1-4, Proceedings*, pp. 123-130, 2003.

[8] F. V. ABREU, *et al.*, "Cellular frustration : A new conceptual framework for understanding cell-mediated immune responses" vol. 4163. Berlin, ALLEMAGNE: Springer, 2006.

[9] F. V. de Abreu and P. Mostardinha, "Maximal frustration as an immunological principle," *J R Soc Interface*, vol. 6, pp. 321-34, Mar 6 2009.

[10] ALMEIDA, *et al.*,"Dynamical instabilities lead to sympatric speciation" vol. 5. Tucson, AZ, ETATS-UNIS: Evolutionary ecology research, 2003.

[11] J. K. Percus, *et al.*, "Predicting the Size of the T-Cell Receptor and Antibody Combining Region from Consideration of Efficient Self Nonself Discrimination," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 90, pp. 1691-1695, Mar 1 1993.

[12] J. E. Gentle, "Random number generation and Monte Carlo methods", 2nd ed. New York: Springer, 2003.

[13] D. Kirk and W.-m. Hwu,"Programming massively parallel processors hands-on with CUDA." Burlington, MA: Morgan Kaufmann Publishers, 2010.

[14] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming." Upper Saddle River, NJ: Addison-Wesley, 2011.

[15] W.-m. Hwu, "GPU computing gems." Amsterdam ; Burlington, MA: Elsevier, 2011.

[16] H. Nguyen and NVIDIA Corporation., "GPU gems 3." Upper Saddle River, NJ: Addison-Wesley, 2008.

[17] N. Farooqui, *et al.*, "A framework for dynamically instrumenting GPU compute applications within GPU Ocelot," presented at the Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, Newport Beach, California, 2011.

[18] F. Macian, *et al.*, "T-cell anergy," *Curr Opin Immunol*, vol. 16, pp. 209-16, Apr 2004.