Redes são dinâmicas!!

Alexandre Domingues Duhau Laborde ***Nº:*** *79448*
Bernardo Lopo Tavares Fernandes ***Nº:*** *70209*
Pedro Filipe Flores Cristovão ***Nº:*** *70273*

Unidade Curricular: Redes Complexas

Prof. Francisco Correia dos Santos
Prof. Alexandre Francisco

Universidade de Lisboa
Técnico, Campus Alameda

Capítulo 1

Introdução

Este é o segundo de dois projectos para a cadeira de Redes Complexas, unidade curricular leccionada no Técnico Lisboa para, entre outros, o Mestrado em Engenharia Informática e de Computadores e Mestrado em Matemática e Aplicações. Foram propostos diversos temas para explorar e, o nosso grupo optou por explorar a implementação do algoritmo epidémico agregado com uma condição que tem influência na topologia da rede.

1.1 Objectivos do Trabalho

Pretende-se com este trabalho implementar um algoritmo epidémico com actualização topológica da rede de acordo com parâmetros do modelo.

Estudar outras implementações do modelo, relativamente à localização espacial das ocorrências.

Entender de que forma a aplicação da função respectiva a um modelo epidémico a um grafo dinânico tem impacto nas conclusões retiradas.

1.2 Conteúdo introdutório

Para alcançar os objectivos procurámos aprofundar um pouco os nossos conhecimentos adquiridos nas aulas, de acordo com os documentos facultados e pesquisa autónoma.

1.2.1 Dinâmicas de Dispersão

Um dos problemas que a activa comunidade que se debruça sobre problemas de redes vai ao encontro das dinâmicas de dispressão, isto é, a forma como decorre propagação em redes complexas e/ou com determindas propriedades. Com o estudo destas dinâmicas houve a necessidade de estabelecer determinados modelos de dispersão, entre os mais tradicionais o *SI*, *SIS* , *SIR* e *SIRS*.

A sigla *SI* significa *Susceptible* e *Infected* e modela doenças que, uma vez alguém seja infectado, não há forma de voltar a um estado susceptível. Exemplos são doenças como a SIDA ou herpes. As restantes letras nos restantes modelos representam *Susceptible* e *Recovered* (ou *Removed*) onde servem de exemplos de doenças, respectivamente, a constipação e a varicela. Em todos estes modelos existem parâmetros β e δ que representam respectivamente uma taxa de infecção ou recuperação. Estas variáveis são determinadas pela doença que o modelo representa.

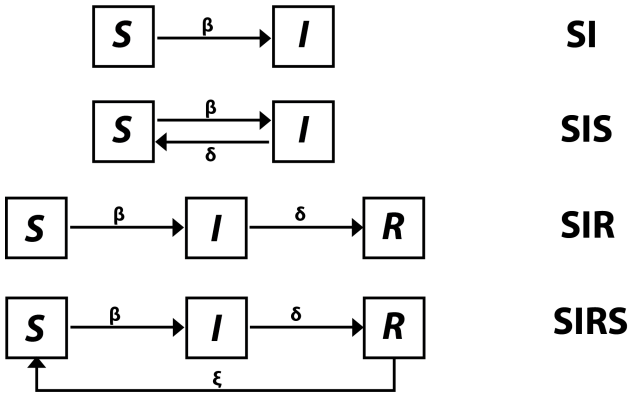

Figura 1.1: Representação dos diversos modelos epidémicos referidos no parágrafo anterior

Modelo SI

O modelo ***SI*** conta com um parâmetro. A fracção de individuos infectados é dada por:

$$x = \frac{I(t)}{N}$$

sendo N o tamanho da população e $I(t)$ o número de indivídos infectados no tempo t. Considerando a infecção temos que a variação vai ser dada por:

$$\dot{x} = x(1-x)\beta < k >$$

para k igual ao número médio de ligações por indivídos. A função de infectados terá aproximadamente o seguinte aspecto:

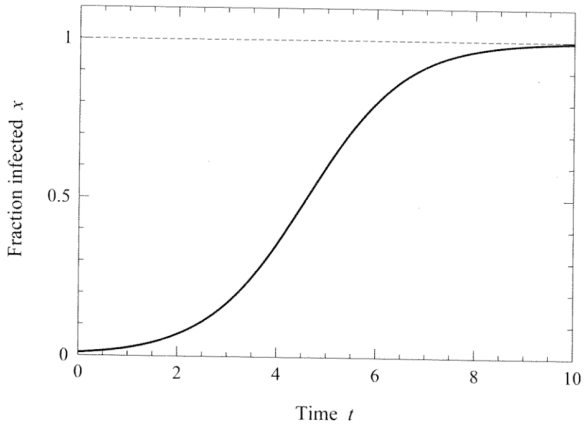

Figura 1.2: Representação do rácio de individuos infectados ao longo do tempo t

Modelo SIS

O modelo ***SIS*** é semelhante ao *SI*, tendo em consideração uma taxa de recuperação. Assim, avariação de x será dada por:

$$\dot{x} = x(1-x)\beta < k > -\delta x$$

E o gráfico terá o mesmo aspecto que o do ***SI***, sendo a única excepção o valor máximo da fracção de pessoas infectadas, pois existem constantemente pessoas a curarem-se e a ficarem doentes.

Modelo SIR

Quanto ao modelo ***SIR*** há mais um estado a ter em conta, no entanto as fracções de estado vão também convergir. A variação de x vai ser agora dada por:

$$\dot{x} = x(1-x-r)\beta < k > -\delta x$$

E os gráficos das fracções de estado terão este aspecto:

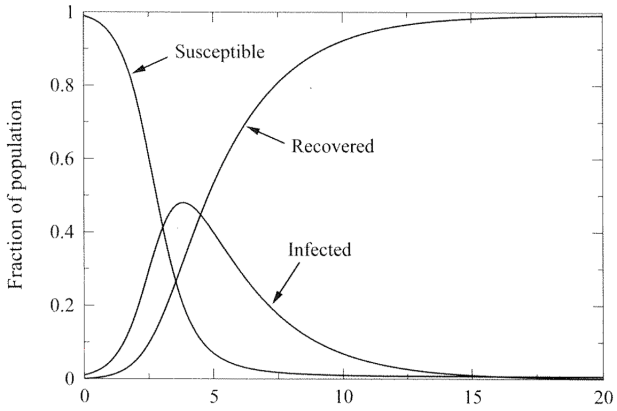

Figura 1.3: Representação dos diferentes rácios de individuos para cada estado ao longo do tempo

Modelo SIRS

Mais uma vez, a variação será dada por:

$$\dot{x} = x(1-x-r)\beta < k > -(\delta + \epsilon)x$$

E o grafico será semelhante ao da Figura 1.2. A fracção em que a população irá estabilizar irá depender dos valores para β, δ e ϵ.

1.3 Esboço da Implementação

Assim que o tema do projecto feoi definido, o grupo começou por esboçar uma ideia de como seria o simulador que iriamos implementar. O simulador consistiria em duas metades em duas *threads*, a primeira, responsável pelo controle da simulação e outra por um engine que desenhará a nossa rede. No algoritmo consta uma *Cadeia de Acontecimentos Pendentes* ***CAP*** e funções independentes que são chamadas pela ordem que o eventos acontecem na CAP. O *selector* selecciona as funções para uma determinada run e o *Starter* serve para configurar a simulação desejada.

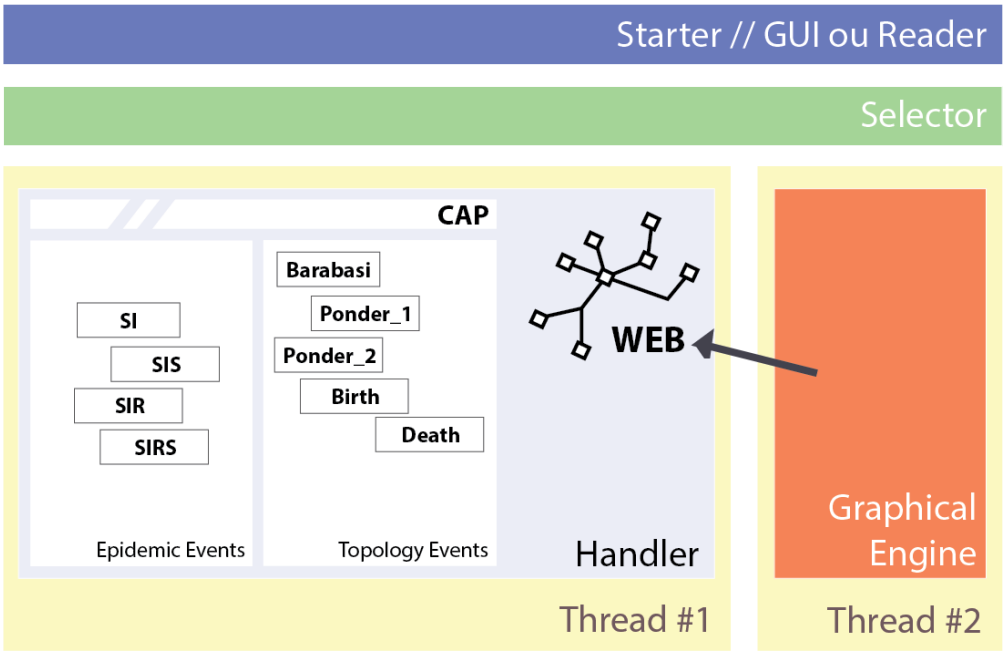

Figura 1.4: Representação *high level* do programa

1.3.1 Descrição da Implementação Dinâmica

Um dos objetivos do trabalho é implementar os modelos debatidos na última subsecção em grafos dinâmicos. Para isso definimos um esquema de implementação de grafos dinâmicos e é disso que iremos falar nesta subsecção.

Em vez de uma implementação discreta, uma abordagem mais comum, decidimos implementar o algoritmo com uma *cadeia de acontecimentos pendentes* (***CAP***). Esta cadeia consiste numa *array* com pares ordenados

$$\{Evento, tempo, vertice\}$$

gerados de acordo com uma heurística fixa. Correr o algoritmo consiste em ir buscar à ***CAP*** o evento com menor tempo associado e colocar ou não um evento igual para um tempo futuro.

Nos eventos estarão implementadas as funções que farão o nosso programa realizar a simulação desejada. Num debate inicial, a equipa decidiu que pelo menos teriamos de ter implementados os seguintes eventos:

- Eventos Topológicos
  - ***Barabasi***: Algoritmo para geração de uma *scale-free network* de acordo com o Barabasi-Albert Model.
  - ***Birth***: Adição de vertices na rede
  - ***Death***: Morte de vértices da rede e suas respectivas adjacências.
  - ***Ponder_1***: Esta função serve para ponderar novas ligações de um determinado vértice a outros arbitrários a que não está ligado e reponderar ligações já existentes. Esta ponderação tomará em consideração o estado de saúde dos vértices envolvidos.
  - ***Ponder_2***: A diferença para o ***Ponder_1*** é que apenas reconsidera novas ligações com vértices a uma distância dentro de um intervalo gerado aleatóriamente. Achamos que

este método é mais descritivo da vida real por haver maior probabilidade em conhecermos amigos de amigos que pessoas escolhidas abritáriamente.

- Eventos Epidemiológico: Implementação da propagação e mudança de estado dos modelos **SI**, **SIS**, **SIR**, **SIRS** para um vértice e seus vértices adjacentes. Definiu-se uma função para cada modelo.

A condição de paragem do algoritmo será um *tempo final* dado como input do programa.

1.3.2 Estrutura de Dados

Para este projecto definimos um grafo ($G = (V, E)$) como um mapa que recebe vértices e devolve a lista de adjacência dos mesmos, isto é:

$$G\ grafo\ :\ G : V \longrightarrow 2^V$$

$$tal\ que\ G(v) = A\ e\ \forall x \in A : \{v, x\} \in E$$

Esta implementação é vantajosa porque tem complexidade $O(1)$ por ser um *hashmap*. É fácil adicionar e remover nós e é bastante flexível.

Os vértices têm também informação sobre o seu estado relativo aos modelos de propagação das doenças outras variáveis como e posição e velocidade, necessárias para a sua representação no *graphical engine*.

1.4 Motivações para a implementação

Grande parte do pensamento por trás da nossa arquitectura provisória para o programa debruça-se sobre expansibilidade. A forma como pretendemos implementar o código, discutida anteriormente, permite uma fácil integração de oturas funções a actuar sobre o grafo. Isto é, se quisermos implementar uma doença com, por exemplo, 6 estados diferentes, o simulador está preparado para o receber e calcular os respectivos rácios sem que tenha de ser alterado qualquer código.

1.5 Ferramentas

Acordou-se que este projecto seria desenvolvido em Java e como tal, utilizámos o ***InteliJ*** IDE para desenvolver o código, fazer debug e correr o programa. Para análise de resultados, *plot* de gráficos e cálculos após o *run* do simulador foi utilizado ***Wolfram Mathematica 10***.

Capítulo 2

Implementação e Programa

2.1 Implementação

Após uma análise mais cuidada do esboço original do programa, a estrutura de dados, e algums dos métodos que tinhamos acordado implementar tiveram de ser ligeiramente adaptados aos nossos objectivos. Assim, vamos falar brevemente das alterações face à ideia que apresentámos na introdução.

Poderá aceder ao código no seguinte link:

$$https://drive.google.com/file/d/0B9FHvuvi9SkgV2FqYWVUeXZ0Sjg/view$$

2.1.1 Eventos

Todos os eventos passaram agora a ser executados para vértices arbitrários passando a ser definidos por

$$\{Evento, tempo\}$$

.

Isto facilita a gestão da ***CAP*** em deterimento de algum controlo (não mais do que parece porque a ordem dos eventos na ***CAP*** também é aleatória). Após um evento ter sido invocado e este ter feito a sua acção atraves do método ***run()***, o evento é responsável por criar outro evento do mesmo tipo que vai ser adicionado à CAP para acontecer no futuro. Isto é feito usando o método ***copy()***. Através da análise da Figura2.2 é possivél perceber que existem 5 tipos de eventos (apesar de poderem ser adicionados quantos o utilizador achar necessários) divididos em três grupos. Esses eventos estão então relacionados com a criação do grafo (verde), com alterações na topologia do grafo (azul) e relacionados com o modo pelo qual a doença e actualizada pelo grafo (amarelo). Este ultimo foi criado com o intuito de poder ter mais controle sobre a doença mas agora, no fim da implementação o grupo começou a ponderar sobre se esta tinha sido a maneira mais correcta de fazer as coisas.

Eventos de Geração

Este tipo de eventos cria grafo que vai ser sobre o qual a simulação da doença vai correr. No decorrer do projecto, foram implementadas três maneiras de um grafo poder ser gerado. No entanto, uma vez que a nossa implementação tem por base a expansibilidade e personalização o utilizador pode adicionar os métodos de geração de grafo que quiser, para isso basta implentar o seu método ***generateGraph()***.

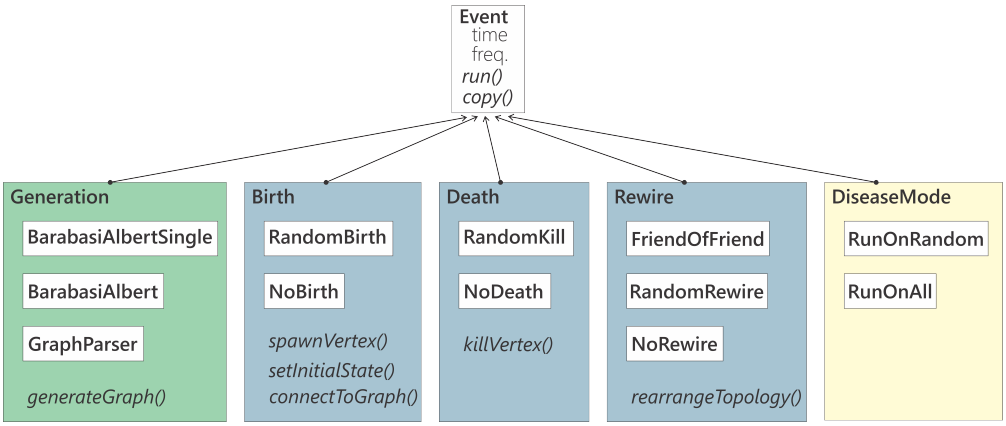

Figura 2.1: Representação dos eventos e seus métodos

Os três métodos implementados pelo grupo foram:

- *BarabasiAlbertSingle* que gera um grafo arbitrário com lei de potências usando o algoritmo de Barabasi-Albert em que a adição de cada nó é um evento independente. Isto torna o processo de geração mais apelativo para visualização.
- *BarabasiAlbert* que é igual ao anterior mas que gera todo o grafo num único evento. Durante o desenvolvimento, este metodo foi abandonado pelo que não recomendamos a sua utilização pois contém alguns erros.
- *GraphParser* que importa um grafo a partir de um ficheiro de texto contendo os nós e as suas arestas.

Eventos Topológicos

Como se pode constatar pela analise da FIGURA X existem três tipos de eventos que afectam a topologia do grafo. Eventos do tipo ***Birth*** adicionam nós ao grafo. Eventos de do tipo ***Death*** removem nós do grafo e eventos do tipo ***Rewire*** alteram as arestas de um dado nó.

Eventos Birth

De modo a poder ter mais controle sobre a maneira como nós novos são adicionados ao grafo, este tipo de evento divide-se em três etapas que o utilizador pode personalizar a seu gosto de modo a poder ter uma representação mais fiavél de como a doença a ser modelada afecta os nascimentos. A primeira etapa controlada pelo método ***spawnVertex()*** controla a geração de vertices, novos, uma vez gerado o novo vértice, ele passa pelo método ***setInitialState()*** que define qual vai ser o seu estado inicial. Tendo agora obtido um novo vértice, o método ***connectToGraph()*** encarrega-se de manipular como é que o vértice se vai juntar ao grafo. Mais uma vez, e a semelhança do resto dos eventos, o utilizador pode criar a sua maneira de adicionar nós ao grafo implementando estes três métodos.

No decorrer deste projecto, o grupo implementou os seguintes modos de adição de nós:

- *NoBirth* que não adiciona nós ao grafo
- *RandomBirth* que liga um nó ao grafo, com um estado aleatório e depois liga o ao grafo usando as regras do Barabasi-Albert Model.

Eventos Death

Como dito anteriormente, este tipo de eventos, removem nós do grafo. O utilizador modelar tal tipo de evento a seu gosto implmentando o método ***killVertex()***. Apesar de nos primeiros momentos deste projecto a equipa ter decidido poder remover nós do grafos x iterações após este ter sido adicionado ao grafo simulando um tempo médio de vida, tal não foi possivel implmentar dadas as nossas restrições temporais.

Como tal a equipa teve de pensar noutras soluções e implementou os seguintes modos de controlar a remoção de vertices:

- *NoBirth* que simplesmente não remove nós do grafo.
- *RandomKill* que escolhe um nó aleatório no grafo e dada uma certa probabilidade remove-o do grafo.

Eventos Rewire

Uma das ideias principais que tivemos quando começamos a debater este projecto foi, tentar perceber como o facto de aplicar regras que fazem a topologia do grafo mudar, teêm influência na capaciadade de uma certa doença se propagar. Neste contexto, um dos eventos que nos pareceu poder vir a ser mais influentes são o tipo de eventos que fazem com que nós distantes se possam ligar. Este tipo de evento sãos modelados pelos eventos Rewire. Podemos obter um evento deste tipo, através da implementação do método ***rearrangeTopology()***. A equipa tentou cumprir as metas que estabeleceu para este evento e implementou os seguintes modos de controlar a topologia do grafo:

- *NoRewire* que simplesmente não altera o grafo.
- *RandomRewire* que dadas as probabilidades de dois nós se ligarem e desligarem, conecta nós aleatórios e remove vizinhos aleatoriamente.
- *FriendOfFriend* A idea por trás desta implementação e o método Ponder2 que no qual faz mais sentido um nó x ligar-se a outro nó y que já se encontra ligado a algum nó z pertencente à vinhizança do nó x simulando assim a maior probabilidade de uma pessoa vir a conhecer um amigo de um amigo do que uma pessoa completamente aleatória. Apesar de na ideia original, este método poder ir buscar ligações novas a qualquer distancia, optámos por limitar esta distancia a 2 e adicionar o estado de saúde dos nós na ponderação sobre a ligação. Este tipo de evento é controlavél através de 5 paramêtros que representam as probabilidades de um nó se ligar a um nó aletório, ligar a um nó saudavél, ligar a um nó infectado e desligar de um nó saudavél, desligar de um nó infectado.

Eventos Simulação da Doença

Estes eventos têm influência sobre como a simulação da doença se propaga pelos nós. A equipa desenhou dois modos da doença se propagar. O modo *RunOnRandom* simula a doença para um nó escolhido aletoriamente, e o modo *RunOnAll* itera o grafo simulando a doença para todos os nós simultaneamente. Apesar de a equipa só ter implementado estes dois modos, se o utilizdor quiser outro modo ele pode adicionalo à aplicação facilmente.

2.2 Doença

Partindo da ideia original de criar um programa completamente e expansivél, a equipa decidiu modelar as doenças como uma máquina de estados. Uma doença ***Disease*** é um conjunto de estados ***DiseaseState*** sendo que na nossa implmentação, não existe qualquer limitação quanto ao numero de estados que uma doença pode ter. Para criar uma doença nova, um utilizador tem apenas de definir o que acontece em cada um dos *DiseaseStates* indicando a alteração do estado do nó a ser processado. Opcionalmente, o utilizador também pode definir uma cor para aquele estado da doença. Essa cor irá ser usada nas representações gráficas tanto do grafo como da estatistica e se o utlizador não definir nenhuma o programa atribuirá uma aleatoriamente.

Como nos tinhamos comprometido antes, o grupo modelou quatro tipos genéricos de doenças *SI, SIS, SIR, SIRS* nos quais a transição entre estados é dada apenas por uma probabilidade controlavél referente a esse estado.

Para demonstar as capacidades da nossa aplicação, o grupo optou também por modelar uma doença em expecifico , a varicela, na qual um nó pode ficar infectado não só a custa do contacto com membros infectados mas também com membros já recuparados que estão em contacto com membros infectados, e que deste modo agem como vectores de contágio da doença. O grupo também começou a trabalhar na modelação de outra doença, a SIDA na qual o nascimento dos nós é afectado pelo estado dos seus progenitores mas devido a limitações temporais, tal modelo não foi concluido.

2.3 Interfaces Gráficas

A equipa também trabalhou em duas interfaces gráficas que permitissem representar o grafo e estatisticas relativamente à simulação. A primeira interface ***Engine*** permite visualizar o grafo. Esta interface corre o *graph embeding* ForceAtlas com parametros controláveis e a cor de cada nó e dada pelo seu estado da doença e o tamanho pelo numero de vizinhos. Nesta interface um utilizador pode navegar pelo grafo com o rato, clicar com o botão direito do rato cria uma forca repulsiva que afasta os nós. O scroll é usado para alterar o zoom do display. A velocidade da simulação pode ser controlada com as setas do teclado e carregar na tecla C faz com que o grafo esteja centrado na imagem.

A segunda interface ***Statistics Plotter*** mostra representação baseada num espaço simplex, dada por um poligono que tem tantos lados como existem estados na doença. Aqui um circulo mostra o ratio entre os vários estados. Esta interface tem os mesmos controles que a anterior com a adicão que permite exportar os dados quando o utilizador pressiona a tecla S que exporta as frações de elementos em ada estado ao longo do tempo, a distribuição de grau e o numero de vértices.

2.4 Manual do Utilizador

A utilização do simulador encontra-se documentada no ficheiro anexado ao relatório. Resta falar sobre o documento *simulation.txt* a que este está a aceder. Falaremos nesta secção de cada uma das linhas do ficheiro.

- **Primeira linha**: Doença, β, δ e ϵ. A função só usa valores que o modelo da doença necessita (e.g.: SI 0.1 0.6 0.2 é o mesmo que SI 0.1)

- **Segunda linha**: Método de propagação da doença. Todos os nós de uma vez ou um nó aleatorio.

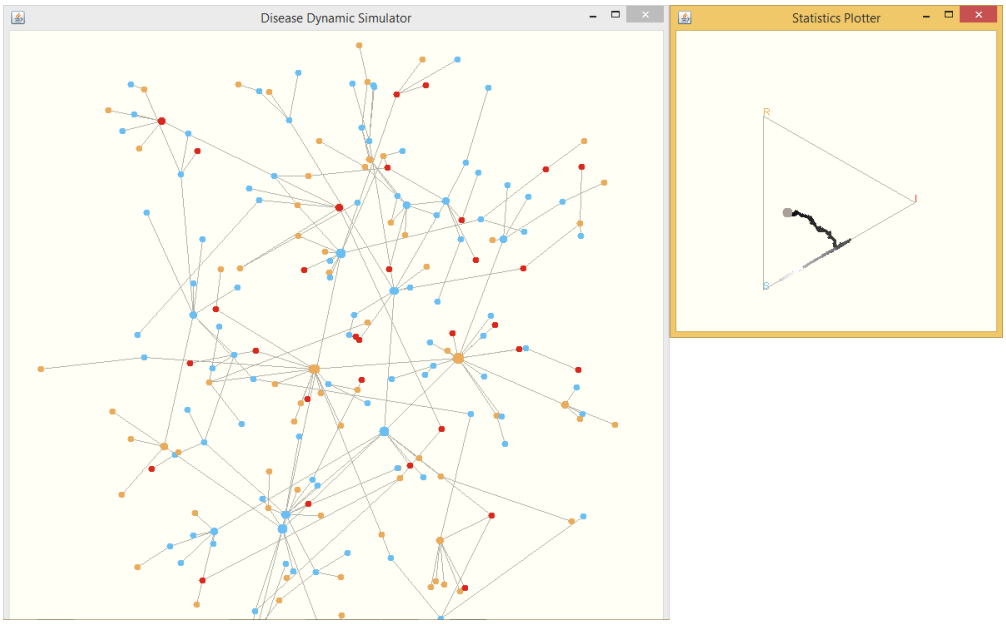

Figura 2.2: Interfaces Gráficas

- **Terceira linha**: Método de geração, número de vértices, probabilidade de nascer infectado.
- **Quarta linha**: Método de nascimento
- **Quinta linha**: Método de morte, parâmetros
- **Sexta linha**: Método de Rewire, Probabailidade de ligar a um nó arbitrário, Probabilidade de um nó saudável ligar a um nó saudável, Probabilidade de um nó saudável ligar a um nó infectado, Probabilidade de um nó infectado se ligar a um nó saudável, Probabilidade de um nó infectado se ligar a um nó infectado.
- **Sétima linha**: engine, On ou Off, parâmetros específicos para modelar as forças relativas ao grafo.
- **Oitava linha**: statistics, On ou Off.

Capítulo 3

Análise de Resultados

Corremos o simulador para os três modelos epidemiológicos com a seguinte configuração. Os dados aqui analizados foram tratados para não repetir valores exactos de um tempo para o outro, mostrando assim alterações apenas quando há alterações de rácio de estado:

Doença 0.1 0.9 0.2

RunOnRandom

BarabasiAlbertSingle 50 3 0.5

RandomBirth

RandomKill 0.01

FriendOfFriend 0.01 0.05 0.01 0.05 0.01

engine On 0.5 20 1 5 0

statistics On

Modelo SI

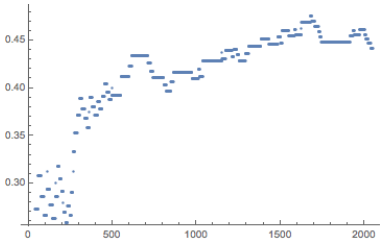

Figura 3.1: Rácio de individuos infectados ao longo do tempo t para o modelo SI

Modelo SIS

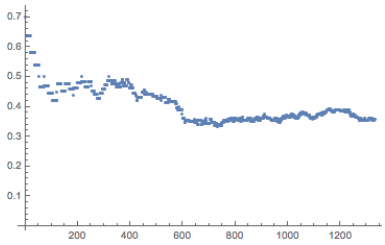

Figura 3.2: Rácio de individuos infectados ao longo do tempo t para um modelo epidemiológico *SIS*

Reparemos que em ambas as simulações anteriores o resultado é aproximadamente o mesmo. Depois de um início com ruído ambas se aprontam a estabilizar. O valor em que este estabiliza é compreensivelmente mais baixo no *SIS*, pois há uma fracção dos vértices que estão a ser tornados susceptíveis no processo.

Modelo SIR

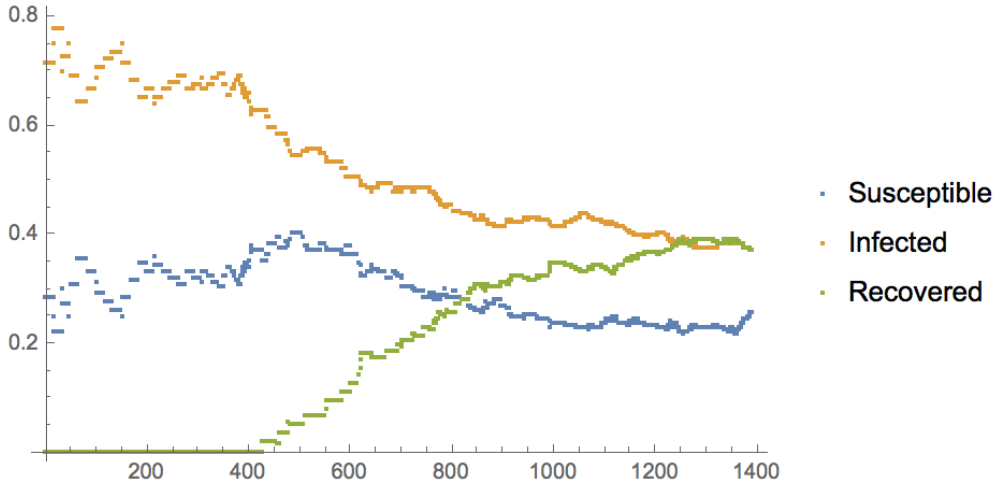

Figura 3.3: Rácio de individuos susceptíveis, infectados e recuperados ao longo do tempo t para um modelo epidemiológico *SIR*

Modelo SIRS

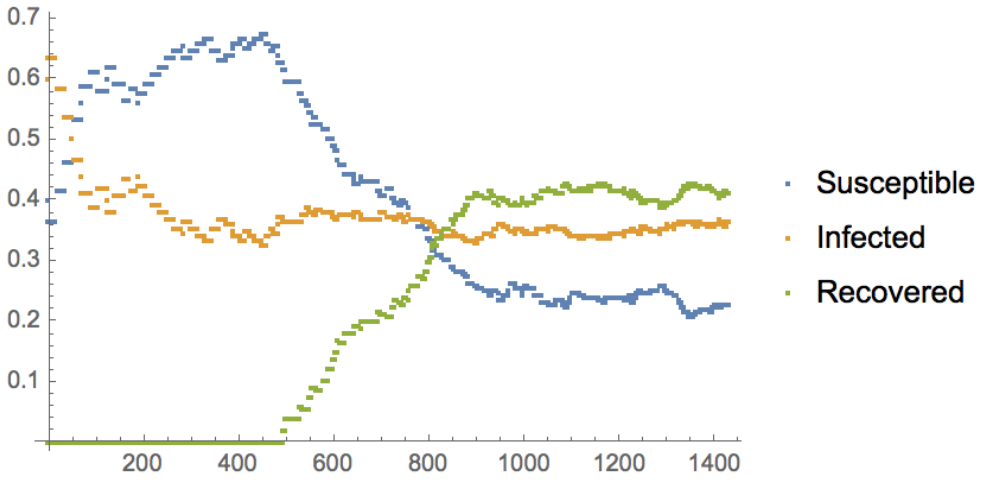

Figura 3.4: Rácio de individuos susceptíveis, infectados e recuperados ao longo do tempo t para um modelo epidemiológico *SIS*

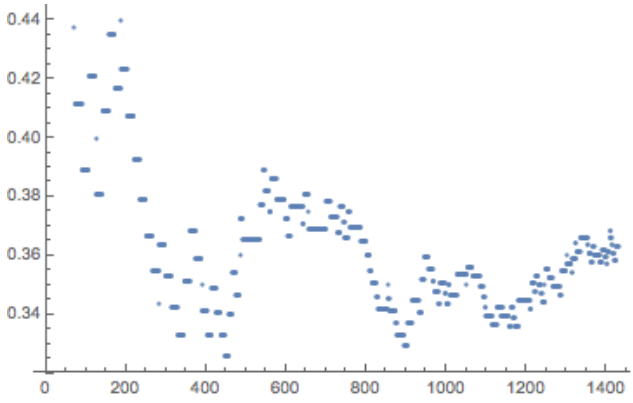

Figura 3.5: Rácio de individuos infectados ao longo do tempo t para um modelo epidemiológico *SIS*

Desta forma é nos possível concluir que o nosso simulador faz convergir os modelos epidemiológicos da forma esperada, de acordo com a Figura 1.3. Porém, há certamente algum ruído nos gráficos que poderá ou não ser justificado pela ordem não determinística de aplicação dos eventos do simulador.

Outra coisa relevante de tomar em consideração é que inicialmente o nosso algoritmo coloca vértices infectados no grafo. Isso faz com que no início não haja a estabilidade que as figuras da introdução retratam.

Capítulo 4

Conclusão

Após a conclusão do nosso trabalho achámos que o simulador, apesar de não conter a tradicional abordagem ao problema de dispersão epidemiológica, consegue de alguma forma ir ao encontro dos resultados esperados. A implementação da ***CAP*** foi relevante para entendermos que, à partida, não deverão haver alterações grandes face aos métodos isentos de permutação de ordem de eventos para as frequências usuais.

No entanto, o grupo de trabalho acredita que para chegar a conclusões mais descritivas do mundo real necessitaria um algoritmo o mais próximo possível do que é propício acontecer na vida real. Achamos que o nosso trabalho é mais um passo nessa direcção e acreditamos que construímos algo capaz de ser actualizado e reforçado para outros problemas e fins.

Assim, convidamos o leitor a experimentar o nosso simulador e até mesmo a editá-lo e a acrescentar novas funcionalidades.