



**Unioeste - Universidade Estadual do Oeste do Paraná**  
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS  
Colegiado de Ciência da Computação  
*Curso de Bacharelado em Ciência da Computação*

**Um Modelo Multiagente *Bitstring* em CUDA para Simular a Propagação de Hipotéticas Doenças Baseadas em Modelagem Compartimental Tipo SEIRS**

*Wesley Luciano Kaizer*

**CASCABEL**  
**2016**

**WESLEY LUCIANO KAIZER**

**UM MODELO MULTIAGENTE *BITSTRING* EM CUDA PARA  
SIMULAR A PROPAGAÇÃO DE HIPOTÉTICAS DOENÇAS BASEADAS  
EM MODELAGEM COMPARTIMENTAL TIPO SEIRS**

Monografia apresentada como requisito parcial  
para obtenção do grau de Bacharel em Ciência da  
Computação, do Centro de Ciências Exatas e Tec-  
nológicas da Universidade Estadual do Oeste do  
Paraná - Campus de Cascavel

Orientador: Prof. Dr. Rogério Luís Rizzi

CASCABEL  
2016

**WESLEY LUCIANO KAIZER**

**UM MODELO MULTIAGENTE *BITSTRING* EM CUDA PARA  
SIMULAR A PROPAGAÇÃO DE HIPOTÉTICAS DOENÇAS BASEADAS  
EM MODELAGEM COMPARTIMENTAL TIPO SEIRS**

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

---

Prof. Dr. Rogério Luís Rizzi (Orientador)  
Colegiado de Matemática, UNIOESTE

---

Profa. Dra. Claudia Brandelero Rizzi  
Colegiado de Ciência da Computação,  
UNIOESTE

---

Prof. Dr. Guilherme Galante  
Colegiado de Ciência da Computação,  
UNIOESTE

Cascavel, 20 de fevereiro de 2017

*You get a shiver in the dark  
It's raining in the park but meantime  
South of the river you stop and you hold everything  
A band is blowing Dixie double four time  
You feel alright when you hear that music ring ...*

# Listas de Figuras

2.1	Fluxo compartmental no modelo SEIRS.	11
2.2	Exemplo de código-fonte em C de um <i>kernel</i> CUDA	26
3.1	Representação gráfica.	30
3.2	Aproximação em matrizes.	30
3.3	Representação indicial e local da vizinhança de Moore.	31
3.4	Representação <i>bitstring</i> do agente $\chi$ .	34
3.5	Ilustração da saída espacial na quadra 445 gerada pelo SIMULA	48
3.6	Ilustração do gráfico de linha gerado pelo SIMULA	48
3.7	Ilustração da saída espalhamento em um lote	49
3.8	Fluxograma da execução completa do SIMULA.	50
3.9	Ilustração da estrutura matricial empregada para a representação do mapeamento lógico dos lotes	51
3.10	Fluxograma para a operação de movimentação dos indivíduos.	55
3.11	Fluxograma para a operação de contato entre os indivíduos.	55
3.12	Fluxograma para a operação de transição de estados dos indivíduos.	55
4.1	Implementação convencional em CPU	59
4.2	Implementação convencional em GPU	59
4.3	Implementação <i>bitstring</i> em CPU	59
4.4	Implementação <i>bitstring</i> em GPU	59
4.5	Gráficos das quantidades de indivíduos <i>versus</i> tempo de simulação	59
4.6	Gráficos dos erros relativos entre as implementações convencional e <i>bitstring</i> em CPU	61

4.7	Gráficos dos erros relativos entre as implementações convencional e <i>bitstring</i> em GPU. . . . .	61
4.8	Gráficos dos erros relativos entre as implementações convencionais em CPU e GPU. . . . .	62
4.9	Gráficos dos erros relativos entre as implementações <i>bitstring</i> em CPU e GPU. . . . .	62
4.10	Dinâmica espaço-temporal para a implementação convencional. . . . .	64
4.11	Dinâmica espaço-temporal para a implementação <i>bitstring</i> . . . . .	65

# **Lista de Tabelas**

2.1	Tabela verdade para a operação unária de negação. . . . .	21
2.2	Tabela verdade para a operação binária de conjunção. . . . .	21
2.3	Tabela verdade para a operação binária de disjunção inclusiva. . . . .	22
2.4	Tabela verdade para a operação binária de disjunção exclusiva. . . . .	22
3.1	Tabela das variáveis relacionadas aos campos dos agentes. . . . .	44
4.1	Parâmetros utilizados nas simulações executadas. . . . .	57
4.2	Dimensões das matrizes utilizadas para representação dos lotes. . . . .	57
4.3	Tempo de execução e <i>speedups</i> obtidos para as diferentes implementações realizadas. . . . .	58

# **Lista de Abreviaturas e Siglas**

SIS	Modelo Suscetível-Infectado-Suscetível
SIR	Modelo Suscetível-Infectado-Recuperado
SEIR	Modelo Suscetível-Exposto-Infectado-Recuperado
SEIRS	Modelo Suscetível-Exposto-Infectado-Recuperado-Suscetível
API	<i>Application Programming Interface</i>
CUDA	<i>Compute Unified Device Architecture</i>
CPU	<i>Central Processing Unit</i>
GPU	<i>Graphics Processing Unit</i>
GPGPU	<i>General Purpose Graphics Processing Unit</i>
EDO	Equação Diferencial Ordinária
PVI	Problema de Valor Inicial
AC	Autômato Celular
LGCA	<i>Lattice Gas Cellular Automata</i>
OpenMP	<i>Open Multi-Processing</i>
MPI	<i>Message Passing Interface</i>
SQL	<i>Structured Query Language</i>
SGBDOR	Sistema Gerenciador de Banco de Dados Objeto Relacional
SIMT	<i>Single Instruction, Multiple Thread</i>
STL	<i>Standard Template Library</i>

# Listas de Símbolos

$\beta$	Taxa de infecção
$\gamma$	Período de exposição
$\alpha$	Período de infectância
$\delta$	Período de recuperação
$\chi$	Agente computacional
$\lambda$	Operador de evolução espaço-temporal
$\mu$	Operador de movimentação
$\rho$	Operador de contato
$\sigma$	Operador de transição de estado

# Sumário

<b>Lista de Figuras</b>	v
<b>Lista de Tabelas</b>	vii
<b>Lista de Abreviaturas e Siglas</b>	viii
<b>Lista de Símbolos</b>	ix
<b>Resumo</b>	xii
<b>1 Introdução</b>	1
1.1 Objetivos . . . . .	2
1.2 Motivação e Justificativas . . . . .	3
1.3 Organização do Trabalho . . . . .	3
<b>2 Fundamentos</b>	5
2.1 Introdução . . . . .	5
2.2 Epidemiologia Matemática e Computacional . . . . .	5
2.3 Modelos compartimentais: Tipos e Classificações . . . . .	9
2.4 Agentes Inteligentes . . . . .	14
2.5 Metodologia de <i>Bitstring</i> . . . . .	20
2.6 <i>Compute Unified Device Architecture - CUDA</i> . . . . .	23
2.7 Trabalhos Relacionados . . . . .	26
<b>3 Metodologias e Modelagem Computacional</b>	29
3.1 Introdução . . . . .	29
3.2 Modelagem do Ambiente de Simulação . . . . .	29
3.3 Modelagem em Operadores aos Agentes . . . . .	31
3.4 Modelagem em <i>Bitstring</i> dos Agentes . . . . .	33
3.5 SIMULA . . . . .	44

3.6	Estruturas de Dados e Estratégias de Implementação . . . . .	50
<b>4</b>	<b>Soluções: Resultados e Discussões</b>	<b>56</b>
4.1	Introdução . . . . .	56
4.2	Discussões Qualitativas, Quantitativas, Eficiência e Acurácia . . . . .	56
4.3	Dinâmicas Espaço-Temporais na Quadra 445. . . . .	63
<b>5</b>	<b>Conclusões</b>	<b>66</b>
	<b>Referências Bibliográficas</b>	<b>69</b>

# Resumo

A aplicação de modelagem compartmental na epidemiologia é amplamente estudada, como pode-se observar na extensa literatura disponível. A simulação de dinâmicas epidemiológicas é de particular interesse no estudo, prevenção e controle de doenças transmissíveis. Com base nestas premissas, este trabalho aborda o problema de simulação de hipotéticas doenças baseadas em modelagem compartmental tipo Suscetível-Exposto-Infectado-Recuperado-Suscetível, SEIRS, por meio de sistema multiagente. Para obter uma adequada formulação à simulação em *General Purpose Graphics Processing Unit*, GPGPU, na plataforma *Compute Unified Device Architecture*, CUDA, ao modelo é empregada a metodologia de *bitstring*, que consiste na manipulação direta dos *bits* de uma palavra computacional. Ao ambiente computacional são utilizadas técnicas e *software* especificamente desenvolvido para manipular dados georreferenciados à especificação e composição de um ambiente apropriado à simulação computacional. Como resultado apresenta-se uma aplicação funcional e adequada para simular eventos epidemiológicos em uma região da cidade de Cascavel/PR. A avaliação do modelo proposto é executada por meio da realização de experimentos numérico-computacionais, em que os resultados da execução de simulações nas diferentes implementações realizadas são comparadas entre si, de forma empírica, sobretudo em aspectos quantitativos. Conclui-se, entre outros aspectos, que a aplicação de modelos compartmentais é adequada à modelagem de eventos epidemiológicos, que a aplicação da metodologia *bitstring* à modelagem dos indivíduos é interessante, sobretudo quanto a redução do uso de memória, e que a paralelização do código utilizando a plataforma CUDA reduz substancialmente o tempo de execução das simulações.

**Palavras-chave:** Epidemiologia computacional, Sistema multiagente, Modelos compartimentais, Modelagem *bitstring*, Plataforma Computacional CUDA.

# **Capítulo 1**

## **Introdução**

Epidemiologia pode ser definida como o estudo da frequência, da distribuição e dos estados ou eventos relacionados à ocorrência ou ao espalhamento de doenças transmissíveis e não transmissíveis em populações específicas, e a aplicação dos resultados desses estudos na prevenção e controle dos problemas decorrentes e relacionados com a saúde pública . Modelos computacionais baseados em indivíduos vêm sendo empregados na epidemiologia para estudar a propagação e a transmissão de doenças, que são processos centrais na dinâmica de doenças infecto-contagiosas. Seu uso viabiliza a modelagem de fenômenos de natureza probabilística e da heterogeneidade nas relações entre os indivíduos e o meio, conferindo mais realidade ao modelo estudado. Neste sentido, uma modelagem compartmental pode ser utilizada à definição de modelos mais complexos, tendo como base a subdivisão da população em categorias, em que os indivíduos fluem entre elas de acordo com determinadas taxas e situações, podendo respeitar as características particulares de uma doença de interesse [Medronho et al. 2008].

À implementação de modelos baseados em indivíduos em uma linguagem computacional pode ser relevante utilizar abordagens mais eficientes à codificação do sistema e definir um ambiente apropriado à execução dos experimentos computacionais. Dependendo da dimensão do ambiente, da quantidade de indivíduos e da complexidade das dinâmicas modeladas, é desejável otimizar o tempo de execução dos experimentos, utilizando os recursos computacionais de processamento e armazenamento disponíveis da forma mais eficiente possível.

## 1.1 Objetivos

O objetivo principal deste trabalho é propor, desenvolver, implementar e avaliar um modelo epidemiológico computacional multiagente, com formulação em *bitstring*, para simular computacionalmente a propagação de doenças que possam ser modeladas por modelos compartmentais tipo Suscetível-Exposto-Infectado-Recuperado-Suscetível, SEIRS, utilizando como ambiente um mapeamento georreferenciado de uma região da cidade de Cascavel/PR. A solução computacional do modelo proposto contempla uma implementação utilizando *Compute Unified Device Architecture*, CUDA, para obter alta eficiência computacional por meio de paralelismo de dados. Para alcançar este objetivo é necessária a realização de objetivos mais específicos que contemplam:

1. Revisão bibliográfica nas temáticas pertinentes ao trabalho, incluindo temas como epidemiologia matemática e computacional, modelagem compartmental, agentes inteligentes e sistemas multiagente, formulação *bitstring*, estruturas de dados e plataforma computacional paralela CUDA.
2. Desenvolvimento e implementação de um modelo multiagente em *bitstring*, baseado em formulação compartmental, tendo como ambiente um mapeamento georreferenciado de uma quadra da cidade de Cascavel/PR.
3. Paralelização em nível de dados do sistema multiagente em *bitstring*, utilizando a plataforma computacional paralela CUDA.
4. O emprego e o aperfeiçoamento de uma ferramenta computacional para viabilizar e otimizar as fases de pré-processamento, processamento e pós-processamento da simulação, como as etapas de configuração e visualização dos resultados obtidos, utilizando o ambiente gerado por tal *software*.
5. Realização de experimentos numérico-computacionais, visando avaliar a acurácia da solução implementada, bem como sua eficiência computacional, e demais aspectos inerentes à modelagem e implementação realizadas.

## 1.2 Motivação e Justificativas

Em conformidade com os objetivos estabelecidos, este trabalho é motivado e se justifica por quê:

- O estudo e aplicação de modelos compartimentais em epidemiologia computacional é relevante, pois mostram-se adequados e flexíveis à modelagem de hipotéticas doenças, bem como são amplamente utilizados, como apresentado na literatura técnica consultada.
- O uso de agentes computacionais em simulações viabiliza a modelagem mais realística dos fenômenos epidemiológicos de interesse, pois os indivíduos têm rica e emergente dinâmica, independentemente uns dos outros, além de viabilizar o emprego de metodologia à sua paralelização.
- A abordagem em multiagente com *bitstring* é relativamente nova e relevante [Paixão 2012], viabilizando a modelagem de agentes computacionais de forma concisa e eficiente, otimizando o consumo de memória, simplificando os processos de captura e configuração de atributos dos indivíduos e facilitando a implementação na plataforma CUDA por reduzir significativamente a quantidade de dados nas transferências entre *Central Processing Unit*, CPU, e *Graphics Processing Unit*, GPU, além de permitir o uso de estruturas de dados simplificadas.
- O uso da plataforma CUDA é atrativo por possibilitar a paralelização massiva do sistema implementado, acarretando ganhos de desempenho desejáveis nos experimentos computacionais realizados.
- Por fim, o trabalho realizado é relevante à formação técnica do futuro profissional em Ciência da Computação.

## 1.3 Organização do Trabalho

Este trabalho apresenta a seguinte organização: no Capítulo 2 são apresentadas e discutidas as fundamentações teóricas utilizadas para a realização do trabalho de conclusão de curso, envolvendo temáticas como a epidemiologia matemática e computacional, modelagem compartimental, agentes inteligentes e sistemas multiagente e modelagem *bitstring*. No Capítulo 3

são apresentados os métodos utilizados à modelagem, implementação e testes do sistema multiagente, como estruturas de dados, linguagens, *Application Programming Interfaces*, APIs, e demais *softwares* de apoio. No Capítulo 4 são discutidas as implementações realizadas por meio do uso das técnicas elencadas nos Capítulos 2 e 3, apresentando-se comparações, resultados obtidos por meio da execução de testes, e conclusões sobre as implementações realizadas.

# **Capítulo 2**

## **Fundamentos**

### **2.1 Introdução**

Neste capítulo são apresentados e discutidos os principais tópicos utilizados como fundamentação teórica à realização deste trabalho. Apresenta-se um breve histórico sobre epidemiologia e são discutidos os principais conceitos da epidemiologia matemática e computacional que serviram de motivação à este trabalho. A seguir são discutidos conceitos e definições importantes sobre modelagem compartmental, juntamente com metodologias computacionais que a utilizam para a simulação de eventos epidemiológicos. Em sequência são discutidas fundamentações teóricas relativas à agentes inteligentes, abordando suas classificações, características e tipos de ambientes em que podem estar inseridos. Por fim, apresenta-se a modelagem *bitstring* proposta neste trabalho, discutindo-se inicialmente conceitos básicos necessários à modelagem e os operadores definidos à manipulação das palavras de *bits*, e a revisão bibliográfica realizada à escrita deste trabalho.

### **2.2 Epidemiologia Matemática e Computacional**

Acredita-se que os primeiros estudos epidemiológicos foram realizados na Grécia Antiga por Hipócrates (460-377 a.C.), que investigou diversas epidemias e suas distribuições ambientais. Após sua morte, seus discípulos deram continuidade aos seus estudos, tomando diferentes frentes de trabalho. Em Roma, os trabalhos epidemiológicos foram iniciados por aqueles inspirados em Galeno (201-130 a.C.), um famoso médico grego. Os estudos romanos produziram resultados importantes na área de epidemiologia, destacando-se a introdução dos censos periódicos e o registro compulsório de nascimentos e óbitos. Em 1850 aconteceu na Inglaterra a

*London Epidemiological Society* organizada pela sociedade *Royal Medical Society*, em que participaram diversos profissionais proeminentes da área epidemiológica. Entre eles estava John Snow (1813-1858), considerado por muitos o pai da epidemiologia moderna. John Snow conduziu estudos pioneiros sobre a mortalidade por infecção pós-cirúrgica nos hospitais militares na Guerra da Criméia e comprovou que o cólera era causada pela ingestão de um agente microbiano presente em águas contaminadas por materiais fecais, iniciando assim as discussões sobre os mecanismos de transmissão hídrica [Medronho et al. 2008].

Ocorreu ainda no século XIX um grande desenvolvimento na área de epidemiologia, destacando-se figuras como Major Greenwood (1888-1949), que introduziu o raciocínio estatístico na pesquisa epidemiológica, contribuindo para a epidemiologia experimental; John Ryle (1889-1950), que propôs a sistematização da História Natural das Doenças; e Jerome Cornfield (1921-1979), que desenvolveu os estimadores de risco relativo e introduziu técnicas de regressão na epidemiologia. Nas décadas de 1960 e 1970 ocorreram grandes transformações na epidemiologia, com a introdução dos computadores e desenvolvimento de técnicas de coleta e análise de dados epidemiológicos. Atualmente, a epidemiologia faz uso de abordagens metodológicas inovadoras, como por exemplo, técnicas de *Data Mining* [Oliveira 2001], que contribuem para a modernização de suas bases e objetos de estudo, abrindo possibilidades para a investigação de grandes processos epidemiológicos em grupos populacionais, a análise de grandes quantidades de dados obtidos em campo e a análise dos dados e das modelagens matemáticas e computacionais. O caráter inovador é bastante incentivado dentro da área, seja pela incorporação de novas metodologias de estudo, novos modelos teóricos ou pela ampliação de objetos de estudos e uso de novas tecnologias [Medronho et al. 2008].

No Brasil, durante o século XIX, aconteceram diversos movimentos importantes, com notável contribuição de pesquisadores brasileiros no desenvolvimento da epidemiologia no país. Dentre os mais importantes destacam-se os esforços de Oswaldo Cruz, que trabalhou no saneamento da cidade do Rio de Janeiro, na época capital do país, e no combate de epidemias como a febre amarela, peste bubônica e varíola, que contaram com forte auxílio militar, e Carlos Chagas, que conduziu estudos sobre a malária, conseguindo controlar um surto epidêmico que ocorria na cidade de Itatinga, interior de São Paulo, sendo que seu trabalho é considerado, ainda hoje, referência mundial no combate à doença. Carlos Chagas também descobriu o protozoá-

rio causador da tripanossomíase americana, que posteriormente ficou mundialmente conhecida como a *Doença de Chagas* [Medronho et al. 2008].

Define-se a epidemiologia como o "*estudo da distribuição e dos determinantes dos eventos ou padrões de saúde em populações definidas, e a aplicação deste estudo para controlar problemas de saúde*", sendo que sua principal diferença à medicina clínica é no uso de populações em seus estudos. Esse diferencial justifica-se por meio do objetivo final da epidemiologia, que é o de melhorar o perfil de saúde das populações, e não somente o de seus indivíduos em particular, e que inferências sobre a relação de determinados fatores e a ocorrência de doenças somente podem ser realizadas por meio do estudo de populações [Medronho et al. 2008]. Observando a dificuldade da realização de pesquisas em campo devido aos custos, complexidade de organização e até impossibilidade de execução, é interessante a aplicação de conhecimentos matemáticos na modelagem de eventos e dinâmicas epidemiológicas. Objetiva-se com esta modelagem descrever e estudar mais profundamente essas dinâmicas com base em conhecimentos provenientes da Matemática, pois assim sendo, pode-se empregar formulação e modelagens que são interpretações formais de eventos biológicos e físicos, que independem do particular caso estudado, podendo o estudo ser generalizado às situações análogas desde que se tenha o conhecimento dos particulares parâmetros e valores da doença, sendo possível então realizar a simulação dos modelos propostos em ambientes virtuais sob controle.

A epidemiologia matemática origina-se dessa união entre conhecimentos matemáticos e conhecimentos biológicos e epidemiológicos. Segundo [Yang 2001] ela é fundamentada em hipóteses matemáticas que descrevem aspectos dos fenômenos biológicos e nas interações entre hospedeiros e parasitas, utilizando-se de conhecimentos biológicos sobre o vírus e, também, da interação desse vírus com o hospedeiro ou com o meio ambiente.

A partir destes conhecimentos desenvolvem-se modelos matemáticos que estão em constante evolução, de acordo com os avanços nos campos das ciências médicas e biológicas. Na maioria das vezes, a construção de modelos matemáticos apoiados em hipóteses é um problema de difícil trato, pois estas devem estar fundamentadas e embasadas matematicamente, sendo esse um objetivo complexo e trabalhoso de se alcançado. A epidemiologia matemática tem por finalidade descrever fenômenos observáveis e não-observáveis e estudar mecanismos de intervenção externa ao sistema de interação hospedeiro-parasita, por meio da construção de

modelos que atuam sobre cenários hipotéticos. Assim, a epidemiologia matemática agrupa conhecimentos epidemiológicos e matemáticos no estudo de epidemias. Os métodos científicos de construção e validação de modelos aplicados à epidemiologia matemática viabilizam estudar tanto as situações de equilíbrio quanto as epidêmicas de uma doença ao longo do tempo [Yang 2001], [Daley e Gani 1999].

Em [Daley e Gani 1999] são discutidos, entre outros temas, os tipos de modelos matemáticos desenvolvidos e estudados na epidemiologia matemática. Pode-se subdividir os modelos matemáticos aplicados à epidemiologia em duas grandes classes:

- **Modelos determinísticos:** São modelos em que as populações de indivíduos suscetíveis, infectados e recuperados são descritos como funções de tempo discreto ou de tempo contínuo, por meio de funções diferenciáveis sob parâmetros não aleatórios. Desta forma é possível aplicar operações diferenciais sobre as funções que regem as dinâmicas de infecção, em que não há a inserção de aleatoriedade, sendo que o processo de infecção da doença evolui de acordo como descrito nas suas regras e funções. São amplamente utilizados para descrever a evolução de uma doença em uma população de grandes proporções.
- **Modelos estocásticos:** São modelos em que se considera uma população de indivíduos, em que cada um destes pertence a uma classe. Os indivíduos da população mudam de classe em instantes de tempo e parâmetros aleatórios dentro de um tempo contínuo ou discreto, respeitando-se os limites das faixas de valores admissíveis, biológica ou fisicamente. A aleatoriedade é inerente em processos infecciosos simulados por estes modelos. Eram comumente utilizados em populações de pequeno porte, embora atualmente são aplicados também à modelagem de dinâmicas epidemiológicas em populações de grande porte.

A integração de modelos teóricos e computacionais é de grande importância no desenvolvimento de modelos epidemiológicos, pois viabiliza o estudo individual de cada parâmetro inerente aos modelos em ambientes simulados, auxiliando na calibração das dinâmicas modeladas e no extenso estudo dos processos dinâmicos em grandes populações de indivíduos e áreas demográficas, sem que seja necessária intervenção humana no local de estudo. O uso de recursos computacionais na simulação de propagação de doenças por meio de modelos matemáticos

epidemiológicos é de grande interesse, pois a cada dia novas tecnologias são apresentadas e o desempenho dos computadores aumenta gradativamente, viabilizando que, cada vez mais, processos epidemiológicos complexos possam ser simulados computacionalmente em tempos de execução satisfatórios. A aplicação de conhecimentos e métodos computacionais à modelagem de epidemias também é interessante por viabilizar a modelagem de comportamentos realísticos das dinâmicas, dos indivíduos e dos ambientes simulados, conferindo maior flexibilidade e versatilidade aos modelos por meio da utilização de técnicas conhecidas das áreas de inteligência artificial, estruturas de dados, georreferenciamento e computação paralela, entre outras metodologias.

## 2.3 Modelos compartimentais: Tipos e Classificações

Em [Alves e Gagliardi 2006] divide-se a modelagem biológica em dois grandes grupos: o grupo experimental e o grupo de simulação. No grupo experimental são construídos e estudados matematicamente modelos experimentais, amplamente utilizados na Biologia. O grupo de simulação agrupa os processos e fenômenos que podem ser simulados por meio de ferramentas computacionais. Em geral, a modelagem matemática e computacional é uma alternativa metodológica com o objetivo de entender e manipular mecanismos e processos em objetos de estudo, com um propósito preditivo, no sentido de antecipar comportamentos epidêmicos das doenças modeladas. O cerne principal da modelagem, seja ela matemática ou computacional, é a abstração das características mais importantes de um sistema natural, descrevendo-a em termos de equações matemáticas ou métodos computacionais.

A abordagem usualmente empregada na modelagem de doenças infecto-contagiosas é a de diagrama de blocos ou em compartimentos, conhecida como modelagem compartmental. Nesta modelagem, aplicada à epidemiologia, a população é classificada em categorias, grupos ou compartimentos disjuntos, de acordo com os estados que se queira modelar de uma determinada doença [Alves e Gagliardi 2006]. Os indivíduos então transitam, de um compartimento à outro, a partir de determinadas probabilidades, taxas ou parâmetros, que dependem das características da dinâmica espaço-temporal da doença. Supondo que  $N(t)$  designa a quantidade total de indivíduos num nível de tempo  $t$  e tomando uma modelagem compartmental, os indivíduos que compõem a população podem ser divididos em classes do tipo suscetíveis,  $S(t)$ ; latentes ou

expostos,  $E(t)$ ; infectados,  $I(t)$ ; e recuperados ou removidos,  $R(t)$ , de modo que  $S(t) + E(t) + I(t) + R(t) = N(t)$ , visto a conservatividade na quantidade de indivíduos na população.

Esses possíveis estados em que cada indivíduo pode estar em um determinado instante de tempo, comumente modelados utilizando-se modelagem compartmental, são conceituados como:

- **Suscetível,  $S$ :** Indica que o indivíduo ainda não contraiu a doença e está apto a adquiri-la.
- **Exposto ou latente,  $E$ :** Indica que o indivíduo contraiu a doença, mas ainda não pode transmiti-la a outros por um determinado período.
- **Infectado,  $I$ :** Indica que o indivíduo está infectado com a doença por um determinado período e é capaz de transmiti-la a outros que estejam no estado suscetível.
- **Recuperado ou removido,  $R$ :** Indica que o indivíduo já foi infectado com a doença anteriormente e não pode transmiti-la nem contaminar-se novamente em um determinado período.

Considerando-se os estados relacionados com o processo infecto-contagioso, usuais modelos empregados são:

- **Modelo Suscetível-Infectado-Suscetível, SIS:** Modelo empregado em situações em que a doença não apresenta período de latência após a infecção e não confere imunidade, passando os indivíduos do compartimento infectado para o suscetível.
- **Modelo Suscetível-Infectado-Recuperado, SIR:** Modelo empregado em doenças em que não há período de latência após a infecção e os indivíduos infectados podem recuperar-se e adquirir imunidade permanente.
- **Modelo Suscetível-Exposto-Infectado-Recuperado, SEIR:** Utilizado em doenças com período de latência após a infecção e que conferem imunidade permanente.
- **Modelo Suscetível-Exposto-Infectado-Recuperado-Suscetível, SEIRS:** Utilizado em doenças com período de latência após a infecção e que não conferem imunidade permanente, passando os indivíduos do compartimento recuperado ao suscetível novamente.

Neste trabalho é utilizado como base à modelagem de dinâmicas epidemiológicas o modelo SEIRS, visto que é um modelo mais abrangente por viabilizar a modelagem de quatro estados, podendo ser utilizado na modelagem de um amplo conjunto de epidemias. O fluxo de transição de estado dos indivíduos para o modelo simplificado SEIRS, desconsiderando as taxas de nascimento, morte de indivíduos e de migração, é como mostrado na Figura 2.1.

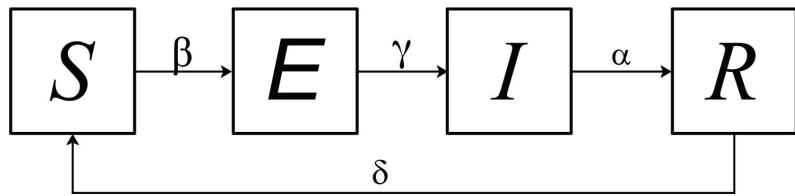


Figura 2.1: Fluxo compartmental no modelo SEIRS.

No diagrama apresentado na Figura 2.1,  $\beta$  designa a taxa de transmissão da doença decorrente do encontro de indivíduos suscetíveis com infectantes. Após ser infectado o indivíduo entra em um estado de latência, permanecendo neste estado por um determinado tempo, designado pela taxa  $\gamma$ . O símbolo  $\alpha$  designa a taxa de recuperação, em que indivíduos infectantes se recuperam da doença, ficando imunes à ela. O taxa de perda de imunidade de indivíduos à doença é caracterizada por  $\delta$ , passando eles novamente ao estado suscetível.

Considerando-se estes modelos compartmentais e suas características, as principais metodologias matemáticas ou computacionais empregadas na sua simulação são:

- **Equações Diferenciais Ordinárias:** Nesta abordagem descreve-se matematicamente o fluxo de transição dos possíveis estados dos indivíduos no modelo SEIRS por meio de um sistema de equações diferenciais ordinárias não lineares, EDOs. A abordagem em EDOs é de particular interesse na modelagem de eventos epidemiológicos, pois possibilita a análise matemática das equações afim de encontrar pontos de equilíbrio e estudar demais propriedades de interesse. Se  $N = S(t) + E(t) + I(t) + R(t)$  designa o tamanho fixo da população de indivíduos, em que suas quantidades são funções do tempo, obtém-se os problemas de valor inicial, PVI, para o modelo compartmental apresentado na Figura 2.1, para o caso contínuo e discreto, como apresentados em 2.1(a) e 2.1(b).

$$\left\{ \begin{array}{l} \frac{dS(t)}{dt} = -\frac{\beta}{N} I(t) S(t) + \delta R(t) \\ S(0) = S_0 \\ \frac{dE(t)}{dt} = +\frac{\beta}{N} I(t) S(t) - \gamma E(t) \\ E(0) = E_0 \\ \frac{dI(t)}{dt} = +\gamma E(t) - \alpha I(t) \\ I(0) = I_0 \\ \frac{dR(t)}{dt} = +\alpha I(t) - \delta R(t) \\ R(0) = R_0 \end{array} \right. \quad \left\{ \begin{array}{l} S^n = S^{n-1} - \frac{\beta}{N} I^{n-1} S^{n-1} + \delta R^{n-1} \\ S^0 = S_0 \\ E^n = E^{n-1} + \frac{\beta}{N} I^{n-1} S^{n-1} - \gamma E(t)^{n-1} \\ E^0 = E_0 \\ I^n = I^{n-1} + \gamma E^{n-1} - \alpha I(t)^{n-1} \\ I^0 = I_0 \\ R^n = R^{n-1} + \alpha I^{n-1} - \delta R^{n-1} \\ R^0 = R_0 \end{array} \right. \quad (2.1)$$

(a)

(b)

Nas formulações apresentadas em 2.1,  $S_0$ ,  $E_0$ ,  $I_0$  e  $R_0$  representam, respectivamente, as distribuições iniciais das populações de suscetíveis, latentes, infectantes e recuperados, e  $n$  indica o nível de tempo do cálculo. Modelos em EDO são bastante utilizados e particularmente interessantes por viabilizarem a realização da análise matemática de suas soluções, o que contribui ao estudo e validação do modelo. Em [Paiva e Nepomuceno 2013] são estudadas questões relativas à estabilidade de um modelo SIR com sistema de vacinação pulsada. Em [Luiz 2012] são discutidos detalhadamente os métodos utilizados à análise matemática de modelos epidemiológicos, sobretudo questões de estabilidade e instabilidade em sistemas não-lineares.

- **Autômatos Celulares:** Autômatos celulares, AC, podem ser classificados como sendo modelos de sistemas dinâmicos discretos no espaço e no tempo, que operam sobre um *lattice* contendo uma quantidade finita ou infinita de células. São caracterizados por interações locais em que cada célula está associada a um estado ou fase por meio de um conjunto discreto de valores. A atualização do estado é realizada a partir dos estados das células vizinhas de acordo com regras locais. ACs possuem um domínio especificado pelo ambiente ou espaço celular em que estes são operados; um espaço de estados das células em que cada célula do *lattice* possui determinado estado; a vizinhança de cada célula; e a regra de transição local que atua sobre a célula, de modo que seu estado pode variar ao ocorrer uma transição. Considerando-se as expressões para o domínio e a regra de transição, e tomando a condição de contorno periódica, o AC para o modelo SEIRS

apresentado na Figura 2.1 é como:

$$\begin{cases} s_{i,j}^{t+1} = s_{i,j}^t - \beta s_{i,j}^t i_{i,j}^t - s_{i,j}^t \left( \sum_{(\alpha,\beta) \in \mathcal{V}^*} \mu_{\alpha,\beta}^{(i,j)} \frac{N_{i+\alpha,j+\beta}}{N_{i,j}} i_{i+\alpha,j+\beta}^t \right) + \delta r_{i,j}^t \\ e_{i,j}^{t+1} = e_{i,j}^t + \beta s_{i,j}^t i_{i,j}^t + s_{i,j}^t \left( \sum_{(\alpha,\beta) \in \mathcal{V}^*} \mu_{\alpha,\beta}^{(i,j)} \frac{N_{i+\alpha,j+\beta}}{N_{i,j}} i_{i+\alpha,j+\beta}^t \right) - \gamma e_{i,j}^t \\ i_{i,j}^{t+1} = i_{i,j}^t + \gamma e_{i,j}^t - \alpha i_{i,j}^t \\ r_{i,j}^{t+1} = r_{i,j}^t + \alpha i_{i,j}^t - \delta r_{i,j}^t \\ s_{i,j}^0 > 0, \quad e_{i,j}^0 \geq 0, \quad i_{i,j}^0 > 0, \quad r_{i,j}^0 \geq 0, \end{cases} \quad (2.2)$$

Em (2.2)  $\beta$  designa a taxa de infecção dos indivíduos suscetíveis por infectantes que ocupam uma mesma posição no *lattice*,  $\mu$  designa a taxa em que indivíduos infectantes transmitem a doença a indivíduos suscetíveis que estejam em posições vizinhas a sua no *lattice*,  $\gamma$  designa a taxa de transição de indivíduos do estado exposto para infectante,  $\alpha$  designa a taxa de recuperação dos indivíduos infectantes e  $\delta$  designa a taxa de perda de imunidade dos indivíduos recuperados.

A solução de (2.2) é obtida por meio da evolução espaço-temporal do AC que fornece os estados finais a partir do cálculo dos estados iniciais. A transição se dá num nível de tempo  $t$  para o consecutivo nível de tempo  $t + 1$ . Em modelagens utilizando ACs é importante adequar os parâmetros da dinâmica afim de garantir a conservatividade na quantidade de indivíduos da população, evitando-se resultados errôneos como o apresentado em [White, Rey e Sanchez 2007].

- **Lattice Gas Cellular Automata:** *Lattice Gas Cellular Automata*, LGCA, são um particular tipo de autômato celular utilizado, por exemplo, na simulação de escoamento e fluxo de fluídios, por meio da modelagem de partículas [McNamara e Zanetti 1988]. Neste modelo, especificamente aplicado à epidemiologia, as partículas são entidades discretas, como os indivíduos da população, em determinados estados, que interagem entre si localmente e se movem no *lattice*. O tratamento dos indivíduos como entidades discretas viabilizam a randomicidade local nas interações e são apropriadas à simulações computacionais. A transmissão da doença ocorre apenas quando um indivíduo suscetível encontra um indivíduo infectante, de forma que a modelagem da movimentação dos indivíduos é de grande importância em modelos deste tipo [Fuks e Lawniczak 2001]. Geralmente são modelados os processos de propagação e colisão. No processo de propagação as partículas são movidas em sua vizinhança à determinadas velocidades, em que somente uma partícula pode

se movimentar em cada conexão entre os elementos na célula do *lattice*. No processo de colisão, regras são utilizadas para determinar o que acontece quando duas ou mais partículas estão em uma mesma posição do *lattice*. Para a construção de um LGCA deve-se especificar um modelo que atenda as características de um autômato celular na fase de propagação e é preciso definir quais regras de colisão que serão utilizadas, que dependem do problema ou da dinâmica que será modelada.

- **Sistemas multiagentes:** Sistemas multiagente consistem na especificação de agentes com objetivos definidos, que interagem entre si e com um ambiente. Na epidemiologia computacional, agentes baseados em modelos são definidos espaço-temporalmente especificando-se como ocorre a transição do seu estado e seu movimento no ambiente ao longo do tempo. A especificação formal de um agente pode ser realizada por meio de um operador de evolução, que é a composição dos operadores de transição temporal, que realiza a transição do estado do agente considerando sua interação com outros agentes e com o ambiente, e de transição espacial, que movimenta os agentes de uma posição à outra dentro do ambiente, considerando seus atributos de conectividade e mobilidade. Sistemas multiagente são interessantes por viabilizarem a modelagem das classes de indivíduos independentemente umas das outras, por sua natureza intrínsecamente paralela, facilitando o desenvolvimento de sistemas multiagentes que podem ser computados em paralelo.

Especificamente neste trabalho, o enfoque é no uso da abordagem de sistemas multiagente à modelagem e implementação do modelo SEIRS descrito anteriormente. Na Seção 2.4 são apresentadas e discutidas em detalhes as características e classificações de agentes e ambientes em que podem estar inseridos, assim como outros conceitos importantes ao tema.

## 2.4 Agentes Inteligentes

Segundo [Russel e Norvig 2003] um agente é uma entidade que, por meio de sensores, pode perceber o meio ambiente em que está inserido, podendo agir sobre este por meio de atuadores, sendo capaz de perceber suas próprias ações, mas nem sempre seus efeitos. É capaz de agir de forma autônoma no ambiente com o intuito de atingir seus objetivos. A exemplo de um agente humano, os olhos, nariz e ouvidos seriam sensores e os braços, mãos e boca seriam os atuadores.

Agentes robóticos podem contar com sensores eletrônicos, como câmeras e infravermelho e como atuadores diversos motores e braços mecânicos. Agentes de *software* podem receber, por exemplo, pacotes de rede e agir enviando outros através da rede. A escolha da ação de um agente em um determinado instante de tempo pode depender de todas as percepções realizadas até aquele momento, e não somente da percepção atual.

O comportamento de um agente pode ser descrito matematicamente por meio de funções, que mapeiam uma ou várias percepções em uma ação. Entre os distintos tipos de agentes estão os agentes racionais que, para cada possível conjunto de percepções, selecionam uma ação que maximize seu desempenho no ambiente, levando em conta seu conhecimento construído ao longo do tempo.

Para a avaliação do sucesso de um agente, é indispensável uma medida de desempenho objetiva, que possa representar claramente como avaliar o desempenho do agente, quando atuando no ambiente. Geralmente, em problemas de otimização, uma função de avaliação é implementada pelos agentes, em que objetiva-se a minimização ou maximização de uma determinada dinâmica ou recurso.

Agentes inteligentes são amplamente utilizados na análise e modelagem matemática e computacional, análise e desenvolvimento de robôs, na área de robótica, em sistemas de extração de informações ou monitoramento de comportamentos ou recursos, em redes de aprendizado e decisão, em jogos eletrônicos, em sistemas comerciais de *marketing* e propaganda, entre diversas outras aplicações.

Um agente, para ser considerado inteligente, deve ser capaz de desempenhar funções autônomas para atingir seus objetivos. Desta forma, quatro conceitos são importantes afim de determinar se um agente é ou não inteligente [Wooldridge 2016] e [Wooldridge e Jennings 1995].

- **Autonomia:** Agentes inteligentes devem ser capazes de agir sem intervenções diretas de humanos ou outros mecanismos externos, tendo controle sobre suas ações e seu estado.
- **Reatividade:** Agentes inteligentes devem ser capazes de perceber seu ambiente e responder, em um tempo hábil, às mudanças que ocorreram, com o intuito de atingir seus objetivos.

- **Pró-atividade:** Agentes inteligentes devem ser capazes de mostrar um comportamento direcionado à objetivos, tomando a iniciativa para alcançá-los.
- **Sociabilidade:** Agentes inteligentes devem ser capazes de interagir com outros agentes para satisfazer seus objetivos.

Um importante componente que interage com o agente é o ambiente. O ambiente pode ser definido como o meio em que os agentes atuam e interagem entre si, sendo que as ações e interações entre os agentes provocam mudanças no ambiente, que são percebidas por estes. Em [Russel e Norvig 2003] e [Wooldridge 2016] são apresentadas diversas classificações para ambientes, destacando-se:

- **Ambiente totalmente observável ou parcialmente observável:** Um ambiente é totalmente observável se os sensores dos agentes que estão inseridos neste ambiente são capazes de prover ao agente o estado completo do ambiente, fornecendo informações relevantes para a escolha de ações. Ambientes totalmente observáveis são interessantes pois os agentes não precisam manter informações internas sobre o estado do ambiente. Um ambiente é parcialmente observável se os sensores dos agentes não são capazes de prover informações do estado do ambiente como um todo, seja por imprecisões ou devido à informações que estão ocultas ou não podem ser obtidas.
- **Determinístico ou estocástico:** Em um ambiente determinístico, seu próximo estado é completamente determinado pelo estado atual e as ações executadas pelos agentes contidos nele. Neste tipo de ambiente, os agentes não precisam se preocupar sobre incertezas, se o ambiente for totalmente observável. Em ambientes estocásticos, o estado futuro do ambiente não depende somente de seu estado atual e das ações de seus agentes, podendo depender de outras informações ou acontecimentos aleatórios. Ambientes parcialmente observáveis geralmente também são ambientes estocásticos.
- **Episódico ou sequencial:** Em um ambiente episódico, as experiências dos agentes são divididas em episódios atômicos, que consistem na percepção do agente e a execução de uma única ação. O episódio futuro não depende de ações ou informações dos episódios passados, sendo que as ações tomadas no episódio atual depende somente dele próprio.

Em ambientes sequenciais as decisões atuais podem afetar as decisões que serão tomadas no futuro. Em geral, ambientes episódicos são muito mais simples do que ambientes sequenciais, pois os agentes não precisam planejar ações futuras.

- **Dinâmico ou estático:** Ambientes que podem mudar de estado enquanto os agentes estão tomando suas decisões de ação são chamados de dinâmicos. O ambiente está constantemente perguntando aos agentes que ações realizarão. Se um agente não decidiu sua ação, o ambiente infere que sua ação é de não realizar nada. Em ambientes estáticos os agentes não precisam se preocupar com o estado atual do meio enquanto decidem qual ação tomar e nem com a passagem do tempo. Ambientes estáticos geralmente são facilmente manipuláveis, sob o ponto de vista dos agentes.
- **Discreto ou contínuo:** A distinção entre ambientes contínuos e discretos pode ser realizada por meio do estado do meio, da forma como o tempo é tratado ou considerado e das percepções e ações dos agentes. Em ambientes discretos, o tempo e seus estados assumem valores finitos e discretos ou são discretizações de informações contínuas, enquanto que em ambientes contínuos, essas informações assumem valores infinitos ou contínuos.
- **Agente único ou multiagente:** Diz respeito ao número de agentes interagindo e atuando no ambiente. Em ambientes de agente único, somente um agente está interagindo e agindo sobre o ambiente, enquanto que em ambientes multiagente, há diversos agentes, que tanto interagem entre si, quanto interagem com o ambiente. Ambientes multiagente podem ainda ser classificados em ambientes competitivos, em que os agentes, em grupos ou individualmente, tomam suas decisões levando em conta a maximização de seu desempenho e a minimização do desempenho de outros, e em ambientes cooperativos, onde os agentes tomam suas decisões tentando maximizar o desempenho de todos os agentes, como um único grupo.

Quanto aos agentes em particular, pode-se classificá-los em quatro tipos básicos, de acordo com suas características e aplicações: [Russel e Norvig 2003], [Wooldridge 2016], [Wooldridge e Jennings 1995].

- **Agentes reativos simples:** Tipo mais simples de agente, com inteligência bastante limitada. São caracterizados pela escolha de ações somente levando em consideração a

percepção atual, ignorando seu histórico de percepções. Sua tomada de decisão é baseada em regras do tipo condição-ação, também chamadas de regras de se-então. A condição consiste na percepção atual que o agente recebe de seus sensores e a ação é atuação do agente de acordo com a percepção. São eficazes em ambientes totalmente observáveis em que as decisões corretas podem ser tomadas partindo das percepções atuais do agente. Agentes reativos simples podem randomizar parte de seu processo de decisão afim de reduzir sua probabilidade de entrada em um estado de repetição infinita de decisões, que pode ser prejudicial em aplicações multiagente.

- **Agentes reativos baseados em modelos:** Agentes especialmente usados em ambientes parcialmente observáveis, que são capazes de manter informações sobre o ambiente, com base em suas percepções, de forma a refletir aspectos atualmente não observáveis do meio. Em geral, esse tipo de agente tem o conhecimento sobre como o ambiente evolui independentemente e como suas ações afetam o meio em que está inserido.
- **Agentes baseados em objetivos:** Nem sempre conhecer o estado atual do ambiente é o suficiente para o agente tomar a decisão correta sobre suas ações. Algumas vezes é necessário que o agente saiba quais estados ou situações são desejáveis. No geral, agentes baseados em objetivos tomam suas decisões levando em conta suas percepções, o conhecimento que adquirem do meio ambiente e seus objetivos. O objetivo pode ser alcançado por simples ações individuais dos agentes ou este pode ter que planejar uma sequência de ações, e pode até mesmo ter que colaborar com outros agentes. Agentes baseados em objetivos são mais flexíveis pois podem ser reconfigurados a qualquer momento de acordo com suas percepções ou mesmo externamente.
- **Agentes baseados em utilidades:** Na maioria dos ambientes, o incorporação de objetivos no agente não é suficiente para modelar comportamentos de alta qualidade. Muitas vezes é necessário agregar diferentes estados não científicos, como sentimentos ou adjetivos. Agentes baseados em utilidades fazem uso de uma função de utilidade que mapeia estados ou percepções em um número real, que está associado ao nível de utilidade do agente. Dependendo do nível de utilidade do agente, este pode decidir por diferentes

ações estando em um determinado estado, com o objetivo de aumentar seu nível de utilidade de acordo com a realização de ações.

Essas classificações podem derivar a uma classe mais abrangente: os agentes cognitivos. Agentes cognitivos podem ser utilizados em ambientes totalmente desconhecidos e tornar-se mais competentes e eficazes por meio do ganho de conhecimento. São capazes de aprender e melhorar suas decisões, por meio das experiências e ações tomadas em tempos passados. Podem medir seu desempenho afim de melhorá-lo no futuro. Apresentam um componente importante: o gerador de problemas, que sugere ações ao agente que podem conduzir à obtenção de novas informações e experiências. Podem também apresentar mecanismos de recompensa ou penalidade, que atuam de acordo com seu desempenho e comportamento. Agentes cognitivos são difíceis de modelar e implementar, por apresentarem comportamentos complexos que mimetizam inteligência, como a de seres humanos ou animais [Russel e Norvig 2003].

A utilização de sistemas multiagentes é interessante pois viabiliza a modelagem e representação computacional dos agentes de diversas formas, sendo atraente compará-las com outras abordagens em aspectos computacionais, como o consumo de memória ou o tempo de processamento. Neste trabalho objetiva-se utilizar a abordagem de sistemas multiagentes, com agentes reativos baseados em modelos, inseridos em um ambiente parcialmente observável, estocástico, episódico, estático e de tempo discreto, para a implementação de dois sistemas que utilizem diferentes formas de representação computacional dos agentes. A primeira forma é aquela utilizada comumente, em que, utilizando recursos providos pela linguagem de programação, agrupa-se os atributos dos agentes em estruturas ou classes de alto nível, fornecendo acesso direto sem nenhuma operação adicional à esses atributos. A segunda forma fará uso de palavras computacionais, como os tipos primitivos das linguagens de programação, com o objetivo de utilizar diretamente seus *bits* no armazenamento, recuperação e configuração dos atributos dos agentes. Na Seção 2.5 são discutidos conceitos basilares à utilização da metodologia proposta para a implementação computacional da segunda forma, definindo-se a organização e as operações em *bits* relativas à descrição e manipulação dos atributos dos agentes.

## 2.5 Metodologia de *Bitstring*

O modelo *bitstring* baseia-se na manipulação direta de *bits* de uma palavra computacional para representar atributos pertencentes aos elementos modelados. A utilização de um modelo *bitstring* justifica-se principalmente pelo fato de que, nesta metodologia, as informações são agrupadas de maneira otimizada, diminuindo o espaço de memória utilizado e, por meio de operações binárias e de deslocamentos, permitirem a rápida manipulação dos dados. Pode-se ainda, dependendo da dimensão do problema, carregar o programa inteiro à memória cache do computador, que é uma memória de alta velocidade localizada próxima à CPU, otimizando ainda mais o processo de execução do programa. São escassas as referências disponíveis na literatura sobre tal metodologia, de modo que a especificação aqui apresentada é fortemente baseada nos trabalhos realizados em [Paixão 2012].

Neste trabalho a aplicação da metodologia de *bitstring* justifica-se pela simplificação que esta traz às estruturas de dados que são utilizadas, sendo que estruturas de dados dinâmicas são de difícil manipulação em arquiteturas tipo GPGPU, e estruturas de dados contíguas em memória são adequadas à arquiteturas deste tipo. Estruturas de dados contíguas em memória, também conhecidas como vetores, são facilmente utilizáveis e suficientes aos propósitos da modelagem *bitstring* à representação computacional dos agentes. Esta metodologia também apresenta uso otimizado da memória, que é determinante no ganho de desempenho na posterior paralelização que foi realizada utilizando a plataforma CUDA. A redução do uso de memória também é importante por diminuir a quantidade de dados que são transferidos entre a CPU e a GPU, reduzindo o tempo à realização de cópias de dados entre os componentes.

Para a implementação de modelos *bitstring* é necessário utilizar uma linguagem de programação que viabilize operações diretas à *bits*. Em diversas linguagens, a exemplo Fortran e C/C++, é possível manipular seus tipos primitivos, *bit* a *bit*, por meio de operações da álgebra booleana, como a negação, conjunção, disjunção e deslocamentos. Os tipos primitivos que podem ser utilizados em operações de *bitstring*, ou seja, que apresentam seus *bits* dispositos de forma contígua na memória, também podem ser chamados de palavras computacionais. É natural, quando da utilização da técnica de *bitstring*, a aplicação de operações em palavras computacionais com o objetivo de manipular seus *bits*. Os *bits* dentro de uma palavra computacional podem ser discriminados quanto à sua posição. Os *bits* mais à esquerda da palavra são

chamados de *bits* mais significativos, enquanto *bits* mais à direita da palavra são chamados de *bits* menos significativos. Esses termos advêm do fato de que, quando no processo de conversão de números binários para decimais, os *bits* mais a esquerda têm maior magnitude do que os *bits* mais a direita. Os tipos primitivos mais interessantes ao uso da técnica de *bitstring* são os inteiros, que geralmente contém 32 *bits*, e os reais, que geralmente contém 64 *bits*. As principais operações binárias que podem ser utilizadas em uma modelagem *bitstring* são explicitadas a seguir. Em [DAGHLIAN 2008] discute-se detalhadamente sobre operações lógicas.

- **Operação de negação:** A operação de negação, também conhecida como NOT, é uma operação unária, ou seja, que é aplicada a somente um operando, que consiste em tomar o complemento de um dígito binário. Se o operando de entrada for 0, será retornado 1 e se for 1 será retornado 0. Sua tabela verdade pode ser expressa como na Tabela 2.1.

Entrada	Saída
0	1
1	0

Tabela 2.1: Tabela verdade para a operação unária de negação.

- **Operação de conjunção:** A operação de conjunção, também conhecida como AND, é uma operação binária, ou seja, que é aplicada a dois operandos, que retorna um *bit* 1 somente se os dois operandos foram 1, e 0 caso contrário. Sua tabela verdade pode ser expressa como na Tabela 2.2.

Entrada 1	Entrada 2	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 2.2: Tabela verdade para a operação binária de conjunção.

- **Operação de disjunção inclusiva:** A operação de disjunção inclusiva, também conhecida como OR, é uma operação binária que retorna um *bit* 1 se ao menos um dos operandos for 1, e 0 caso contrário. Sua tabela verdade pode ser expressa como na Tabela 2.3.

Entrada 1	Entrada 2	Saída
0	0	0
0	1	1
1	0	1
1	1	1

Tabela 2.3: Tabela verdade para a operação binária de disjunção inclusiva.

- **Operação de disjunção exclusiva:** A operação de disjunção exclusiva, também conhecida como XOR, é uma operação binária que retorna um *bit* 1 se somente um dos operandos for 1, e 0 caso contrário. Sua tabela verdade pode ser expressa como na Tabela 2.4.

Entrada 1	Entrada 2	Saída
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 2.4: Tabela verdade para a operação binária de disjunção exclusiva.

- **Operações de deslocamento:** As operações de *shift* ou de deslocamento consistem em operar um conjunto de *bits* em uma palavra computacional em um determinado fator, à esquerda ou à direita. As operações de deslocamento podem ser realizadas de duas maneiras:

– **Deslocamentos não circulares:** São deslocamentos em que os *bits* expelidos de uma extremidade da palavra são descartados, de modo que novos *bits* com valor 0 são inseridos na extremidade oposta para manter seu tamanho constante. Tomando como exemplo uma palavra de 5 *bits*,  $01001_2$ , e aplicando um deslocamento não circular de fator 1 à direita, a palavra resultante do processo é  $00100_2$ . Note que o *bit* menos significativo foi perdido e um *bit* 0 foi inserido na posição mais significativa da palavra.

– **Deslocamentos circulares:** São deslocamentos em que os *bits* expelidos de uma extremidade da palavra são inseridos na extremidade oposta, conservando os *bits* deslocados dentro da palavra. Tomando novamente como exemplo a palavra  $01001_2$ ,

se aplicado um deslocamento circular de fator 1 à direita, a palavra resultante é  $10100_2$ . Note que o *bit* menos significativo, que foi expelido na extremidade à direita da palavra, foi reinserido na extremidade à esquerda da palavra.

Neste trabalho de conclusão de curso propõe-se o uso da metodologia de *bitstring* para a modelagem dos atributos dos agentes constituintes da população de indivíduos no sistema multiagente. Por meio das operações citadas anteriormente, são definidos métodos para a captura e configuração dos atributos dos agentes de forma otimizada, utilizando uma linguagem de programação que seja apropriada às operações aqui descritas.

## 2.6 *Compute Unified Device Architecture - CUDA*

Por muito tempo, o método mais utilizado para o aumento de desempenho em processadores era o aumento da velocidade em que o relógio do processador operava. De fato, muitos dos ganhos de desempenho obtidos durante o desenvolvimento dos processadores foram conseguidos desta forma. Porém, os desenvolvedores de circuitos de processamento foram forçados a procurar alternativas à este método, devido a diversas limitações fundamentais na fabricação de circuitos integrados, como por exemplo, restrições de calor e potência, que estavam se aproximando do limite físico suportado pelos transistores. Assim, a alternativa encontrada foi a de mimetizar o que já ocorria em supercomputadores: extrair desempenho por meio do aumento na quantidade de unidades de processamento por processador, colocando dois, quatro, oito ou mais unidades de processamento em cada processador, movimento que ficou conhecido como a revolução *multicore* [Sanders e Kandrot 2010].

Enquanto os processadores evoluíam por meio do aumento da quantidade de unidades de processamento por processador, as GPUs também passavam por uma revolução. Inicialmente, o objetivo principal do uso de placas gráficas em computadores era a de oferecer ao sistema um *hardware* assistente para processamentos gráficos de imagens. O aumento da demanda das placas gráficas devem-se ao lançamento de jogos eletrônicos, que necessitavam de alto processamento gráfico para garantir fidelidade de imagem. Assim, diversas tecnologias para desenvolvimento de aplicações gráficas foram lançadas, como, por exemplo, as bibliotecas DirectX [DirectX 2016] e OpenGL [OpenGL 2016]. Essas APIs eram as únicas formas que os programadores possuíam para interagir com as placas gráficas, de forma que, diversos pesquisadores

exploravam as APIs gráficas como forma de solucionarem problemas de propósito geral. Embora o desempenho obtido com o uso de APIs gráficas se mostrava interessante e promissor, seu modelo de programação era restritivo e complexo. Com o lançamento de placas gráficas de propósito geral e da biblioteca CUDA, buscava-se incluir componentes especificamente desenvolvidos para a computação em GPU, que poderiam ser facilmente utilizados por programadores para implementar as soluções requeridas. A biblioteca CUDA foi desenvolvida inicialmente para o uso conjunto com a linguagem de programação C, que era utilizada como padrão pela indústria. Atualmente a biblioteca CUDA pode ser utilizada em conjunto com outras linguagens de programação como C++, Fortran, Python e C# [Solutions 2016]. As principais áreas de conhecimento que atualmente fazem uso da API CUDA são as de processamento de imagens médicas, simulação computacional da dinâmica de fluídos e ciência ambiental, sendo a biblioteca bastante utilizada no meio acadêmico [Sanders e Kandrot 2010]. É importante classificar paralelismo, como discutido em [Cook 2013]:

- **Paralelismo baseado em tarefas:** Está relacionado com a execução de múltiplos processos concorrentemente, que podem não ter relação uns com os outros. Um exemplo clássico de paralelismo baseado em tarefas é um sistema operacional, em que diversos processos estão sendo executados ao mesmo tempo. Cada processo pode ser escalonado para um *core* separado na CPU, por exemplo.
- **Paralelismo baseado em dados:** O foco não está nas tarefas que devem ser executadas, mas sim nos dados que precisam ser transformados. Neste tipo de paralelismo, os dados são divididos, quando possível, em partes de igual tamanho e são escalonados para cada *core* da CPU, em que é aplicada uma mesma operação sobre todos os dados.

As GPUs geralmente apresentam uma arquitetura baseada em memória compartilhada, o que significa que todas as *threads* têm acesso a todos os dados de forma igualitária. Por este motivo, o modelo de paralelismo baseado em dados é o modelo efetivamente utilizado em GPUs, com uma abordagem *Single Instruction, Multiple Thread*, SIMT. Neste modelo, o programador define em *kernels* as operações que são executadas pelas *threads*. Os *kernels* são os métodos ou funções que somente podem ser executadas pela GPU. O *kernel* então lê os dados de forma uniforme, aplicando as operações definidas sobre os dados.

Como as GPUs apresentam arquitetura baseada em memória compartilhada, para a implementação de aplicações paralelas em CUDA, utiliza-se um padrão baseado em laços de repetição. Neste padrão, cada *thread* indexa um índice de um laço de repetição, que efetivamente indexa uma porção de memória, em que a *thread* executa o *kernel* independentemente. Assim sendo, a execução das iterações de uma laço de repetição são executados paralelamente sobre os dados, observando que esse comportamento somente é possível se não existirem dependências de dados entre as iterações. Desta forma é possível explorar a localidade espacial dos dados, pois *threads* adjacentes irão acessar dados em posições adjacentes na memória, assim como os blocos acessados por cada *thread* estão em posições contíguas na memória. A localidade espacial é considerada um fator de grande importância à obtenção de ganho de desempenho em aplicações executadas em GPU [Cook 2013].

As unidades de processamento nas GPUs estão organizadas em *grids*, *blocks* e *threads*, sendo um *grid* um conjunto de *blocks* e um *block* um conjunto de *threads*. Devido a esta organização, é possível obter um alto nível de paralelismo de dados, pois os identificadores dos *grids*, *blocks* e *threads* são utilizados para compor um identificador único, que identifica a porção de memória que uma *thread* irá acessar e executar o *kernel* [Cook 2013].

Como exemplo, a Figura 2.2 ilustra um trecho de um código-fonte em C para um *kernel* CUDA que executa uma soma de vetores. Como um identificador único pode ser calculado para cada *thread*, então cada *thread* usa seu identificador para indexar uma posição dos vetores de entrada e saída, que são designados pelas variáveis *a*, *b* e *c*. O identificador de cada *thread* é identificado no código pela variável *threadID*. Como a quantidade de *threads* disponíveis na placa gráfica pode ser maior do que a quantidade de elementos nos vetores, então é necessário ainda testar se o identificador da *thread* é menor do que a quantidade de elementos dos vetores, que é designado pela variável *n*. Assim, evita-se acesso indevido à posições que memória que não pertencem aos vetores. Note que não existe um laço de repetição que percorre todos os elementos dos vetores. Esse laço é eliminado graças a utilização de paralelismo baseado em dados.

```

__global__ void somaVetor(double *a, double *b, double *c, int n) {
    int threadID = blockIdx.x*blockDim.x+threadIdx.x;
    if (threadID < n) {
        c[threadID] = a[threadID] + b[threadID];
    }
}

```

Figura 2.2: Exemplo de código-fonte em C de um *kernel* CUDA

Para que seja possível a execução dos *kernel*s sobre os dados que estão na memória compartilhada da GPU, é necessário realizar a sua cópia, da memória principal do computador para memória compartilhada da GPU. É importante notar que, a GPU não consegue acessar diretamente dados na memória principal do computador, assim como a CPU não consegue acessar diretamente dados da memória da GPU. Essa etapa de cópia de dados é um importante fator limitante no ganho de desempenho em aplicações GPGPU, pois geralmente operações de transferência e cópia de memória entre dispositivos são muito custosas computacionalmente. Assim, é desejável que os programadores que desejam utilizar GPUs para acelerar suas aplicações, projetem suas aplicações de forma a reduzir a quantidade de vezes em que são realizadas as cópias de memória, assim como a quantidade de memória total que precisa ser copiada. Efetivamente, a quantidade de vezes de realização de cópias de memória pode ser reduzido se os dados foram mantidos na GPU pelo máximo de tempo possível, em que a cópia de dados da CPU para a GPU seja realizada uma vez ao início do processamento e a cópia da GPU para CPU seja realizada uma vez ao final do processamento [Cook 2013]. A redução na quantidade total de memória que precisa ser copiada somente pode ser alcançada se forem reduzidos os tamanhos, em *bytes*, das estruturas que guardam as informações que precisam ser processadas pela GPU.

Neste trabalho, objetiva-se utilizar a técnica de *bitstring* como forma de reduzir a quantidade de *bytes* que precisam ser copiadas entre CPU e GPU. Com a aplicação da técnica, espera-se, empiricamente, diminuir o consumo de memória, para o vetor de agentes, em até 80%, observando que, para cada agente, uma implementação convencional utilizaria 20 *bytes* para armazenar os cinco atributos do agente, enquanto que, para o modelo *bitstring* proposto, são necessários somente 4 *bytes*.

## 2.7 Trabalhos Relacionados

Para efeitos de comparação dos resultados obtidos neste trabalho, quando viável, com outros disponíveis na literatura técnica, realizou-se uma revisão bibliográfica em trabalhos que

integrem aplicações executadas em arquiteturas de alto desempenho, com modelagem *bitstring* às aplicações em epidemiologia matemática e computacional. Não foram encontrados muitos que contemplam tais aspectos. Alguns dos mais relevantes e compatíveis com o presente trabalho em desenvolvimento são apresentados em sequência.

Em [Willem 2015] é proposto e implementado um modelo baseado em agentes para a propagação de doenças infecciosas, utilizando diversos modelos compartimentais. O principal objetivo do trabalho foi estudar modelos baseados em agentes para propagação de doenças sob importantes aspectos como a estimativa de parâmetros, os padrões de contato sociais e a eficiência computacional. Afim de melhorar a eficiência computacional da implementação realizada em C++, o autor utilizou a API *Open Multi-Processing*, OpenMP [OpenMP 2016], à paralelização do código.

Em [Aaby, Perumalla e Seal 2010] são estudados, entre outros aspectos, mecanismos para a paralelização de simulações com milhões de agentes baseados em modelos, explorando diferentes arquiteturas paralelas como multi-GPUs e *clusters*, bem como diferentes plataformas disponíveis como o CUDA, *pthreads* e *Message Passing Interface*, MPI [OpenMPI 2016]. Os autores concluem que a solução proposta no trabalho é executada até quatro vezes mais rápida em arquiteturas multi-GPUs.

Em [Lysenko, D’Souza e Rahmani 2007] são apresentados diversos algoritmos paralelizáveis em nível de dados para a simulação de modelos baseados em agentes e é proposto um alocador para a replicação de agentes. Os autores afirmam que os algoritmos apresentados no trabalho podem ser facilmente implementados em arquiteturas GPGPU, viabilizando ganhos de desempenho substanciais, e concluem, entre outras questões, que o ganho de desempenho obtido em arquiteturas GPGPU é interessante, porém a implementação utilizando esta tecnologia ainda é contra-intuitiva.

Em [Holvenstot, Prieto e Doncker] é investigada a efetividade de arquiteturas GPGPU na aceleração da execução de simulações de modelos baseados em agentes, com o objetivo de auxiliar a tomada de decisão em políticas públicas relacionadas à pandemias como a Influenza. Os resultados apresentados no trabalho mostram *speedups* de 94% em simulações implementadas em GPU com até 50 milhões de agentes. Os autores concluem que a implementação realizada é

escalável linearmente no número de pessoas, sendo interessante seu uso no auxílio em tomadas de decisão em tempo real.

Em [Holvenstot 2014] utilizou-se um modelo SIR para simular a propagação de doenças em um modelo baseado em agentes. A implementação do modelo foi realizada utilizando a linguagem de programação C++ e as bibliotecas OpenMP e CUDA foram aplicadas à sua paralelização. Os resultados apresentados no trabalho mostram ganhos de desempenho interessantes, tanto no uso do OpenMP, quanto no uso do CUDA. O autor conclui, entre outros aspectos, que o *speedup* obtido na implementação realizada em GPU aumenta conforme o porte da simulação cresce, até que se atinja os limites de memória da GPU.

Em [Paixão 2012] foi proposto e utilizado um modelo *bitstring* ao estudo e simulação da propagação do vírus da Dengue. O modelo foi implementado utilizando a linguagem de programação Fortran e a manipulação dos *bits* das palavras computacionais foi realizada por meio de funções próprias fornecidas pela linguagem. A respeito da aplicação do modelo *bitstring* na simulação da Dengue, o autor conclui que o modelo proposto viabiliza a redução no uso de memória e de processamento em cerca de até 80%.

# **Capítulo 3**

## **Metodologias e Modelagem Computacional**

### **3.1 Introdução**

Neste capítulo são apresentadas as metodologias computacionais utilizadas à realização deste trabalho. São discutidas as modelagens realizadas à representação do ambiente e dos indivíduos no sistema multiagente proposto, expondo-as em termos de operadores e ao uso da metodologia em *bitstring*, as linguagens de programação e as motivações de seus usos, as estruturas de dados desenvolvidas à apropriada aplicação da modelagem *bitstring* e paralelização em CUDA e estratégias de implementação do sistema multiagente. Em sequência é apresentado o *software* especialmente desenvolvido à manipulação de informações georreferenciadas, que é utilizado como apoio às operações de configuração de parâmetros e do ambiente e à visualização de resultados das simulações que são executadas. Por fim são discutidas questões relativas à API para programação paralela CUDA, apresentando suas principais características e aplicações.

### **3.2 Modelagem do Ambiente de Simulação**

Para a modelagem do ambiente computacional, que servirá à execução de simulações da dinâmica epidemiológica, é utilizada como base uma região geográfica da cidade de Cascavel/PR, mais especificamente a quadra 445, que está localizada no centro da cidade de Cascavel/PR e é delimitada geograficamente pelas ruas Maranhão, Castro Alves, Curitiba e Visconde de Guaporé. Esta quadra foi escolhida por ser um bom modelo geométrico, com relação às outras

quadras da cidade, o que é interessante para desenvolver as estruturas de dados utilizadas na implementação do *software*. As Figuras 3.1 e 3.2 ilustram a quadra em questão.

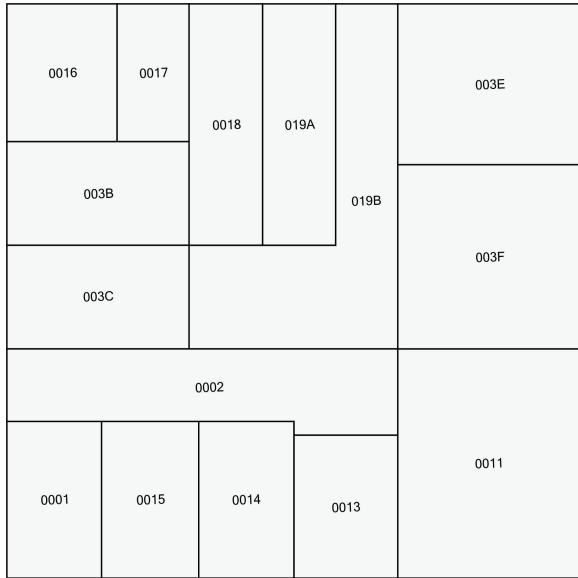


Figura 3.1: Representação gráfica.

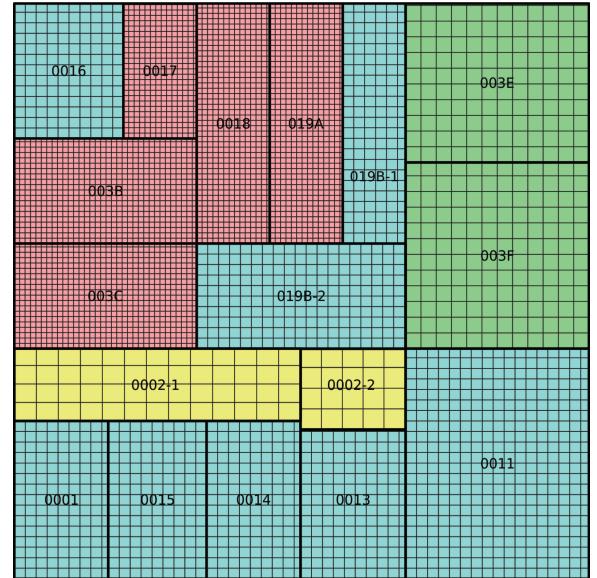


Figura 3.2: Aproximação em matrizes.

A quadra 445 contém 15 lotes, sendo necessário mapear cada um destes para uma estrutura matricial que viabilize sua representação e implementação no sistema de simulação. Para tanto, os lotes 0002 e 019B são divididos em outros dois lotes, pois apresentam geometria irregular de difícil aproximação em estruturas matriciais. Essa divisão é realizada de tal forma em que as matrizes resultantes representem, aproximadamente, a geometria original do lote dividido. Aos outros lotes da quadra 445 não foi aplicada nenhuma operação adicional, por apresentarem geometria regular, que pode ser facilmente aproximada por matrizes. Com as operações de divisão dos lotes irregulares, a quadra 445 que originalmente contém 15 lotes, é mapeada sobre um conjunto com 17 estruturas matriciais, que representam os lotes de forma apropriada à sua implementação computacional.

Como os lotes da quadra 445 são mapeados para estruturas matriciais que viabilizam sua implementação em linguagens de programação, à cada lote estará associada, no sistema de simulação, uma matriz de  $n$  linhas e  $m$  colunas, que representa sua região física. Essa especificação é importante pois à modelagem dos agentes é necessária a descrição do ambiente em que se movem e habitam. Cada agente terá uma determinada posição na quadra, que é completamente especificada pelas coordenadas  $x$  e  $y$  da matriz do lote e por um identificador do lote em

que o agente se encontra. As coordenadas  $x$  e  $y$  representam, respectivamente, a linha e coluna da matriz do lote em que o agente está localizado.

À modelagem das vizinhanças de uma determinada posição  $(x, y)$  de um lote qualquer, é utilizada a vizinhança de Moore, que é amplamente aplicada na modelagem de autômatos celulares. A Figura 3.3 ilustra graficamente os incrementos locais que são realizados às coordenadas de uma posição qualquer para a obtenção de sua vizinhança de Moore.

$(-1, -1)$	$(-1, 0)$	$(-1, 1)$
$(0, -1)$	$(0, 0)$	$(0, 1)$
$(1, -1)$	$(1, 0)$	$(1, 1)$

Figura 3.3: Representação indicial e local da vizinhança de Moore.

Assim sendo, do ponto de vista posicional, a vizinhança de Moore de uma posição  $(x, y)$  qualquer consiste no conjunto das posições  $(x, y + 1)$ ,  $(x, y - 1)$ ,  $(x + 1, y)$ ,  $(x + 1, y + 1)$ ,  $(x + 1, y - 1)$ ,  $(x - 1, y)$ ,  $(x - 1, y + 1)$  e  $(x - 1, y - 1)$ . A definição das vizinhanças de um determinada posição de um lote é relevante às operações de movimentação dos agentes da população, em que ocorre a simulação.

### 3.3 Modelagem em Operadores aos Agentes

Os modelos apresentados na Seção 2.3 serviram à fundamentação do modelo baseado em agentes, que é desenvolvido e utilizado na implementação e execução de simulações computacionais. A modelagem empregada para simular o espalhamento de hipotética doença de transmissão direta em indivíduos considera agentes baseados em modelos, que são definidos espaço-temporalemente especificando-se como ocorre a transição do seu estado num intervalo de tempo e seu movimento no ambiente, de uma posição para outra no passo de tempo. Um passo de tempo é especificado como um ciclo de transição.

Um agente  $\chi(t)$  é definido espaço-temporalemente especificando-se como ocorre a transição do seu estado num intervalo de tempo  $t$  e seu movimento no espaço, que é o ambiente computa-

cional em que o agente é especificado. O estado do agente é especificado por meio do conjunto de atributos, como apresentado em (3.1).

$$\chi(t) \equiv (L, X, Y, C, Q) \quad (3.1)$$

cujos significados dos identificadores dos atributos do estado interno do agente  $\chi(t)$  são como:

- **Lote,  $L$ :** Identificador do lote que o agente  $\chi(t)$  se encontra.
- **Posição em x,  $X$ :** Coordenada  $x$  da posição do agente  $\chi(t)$  no lote.
- **Posição em y,  $Y$ :** Coordenada  $y$  da posição do agente  $\chi(t)$  no lote.
- **Contador de controle,  $C$ :** Contador de ciclos que controla os períodos de transição entre os estados do agente.
- **Estado,  $Q$ :** Identificador do estado atual do agente  $\chi(t)$ .

A especificação formal de um agente é realizada por meio de um operador de evolução que define o estado atual do agente, quando interagindo com o ambiente. Esse operador decorre da composição entre os operadores de transição temporal, que realiza uma transição do estado interno do agente considerando-se sua interação com outros agentes e com o ambiente, e o operador de transição espacial, que movimenta o agente de sua posição para outra, considerando-se os atributos de conectividade e mobilidade. À dinâmica do agente são considerados três tipos de operações:

- **Movimentação:** Nas operações de movimentação, os agentes são movimentados dentro de um ambiente virtual com topologia matricial por meio de suas vizinhanças de Moore às posições escolhidas aleatoriamente, respeitando-se os limites do ambiente.
- **Contato:** Nas operações de contato ocorre, probabilisticamente, a transmissão da doença por meio dos agentes infectados para os agentes suscetíveis que ocupam uma mesma posição no ambiente.
- **Transição de estados:** Nas transições de estados, ocorre a passagem de estados dos agentes de expostos para infectantes, de infectantes para recuperados e de recuperados para suscetíveis.

Tais operações são realizadas na sequência em que foram apresentadas e uma vez a cada ciclo, que consiste na aplicação dos operadores sobre a população de agentes e geração de arquivos de saída. Uma simulação é composta por uma determinada quantidade de ciclos.

Cada agente implementa uma operação de evolução  $\lambda$  que atualiza o estado atual do agente quando interagindo com o ambiente, definido como  $\lambda(\chi(t))$ , que decorre da composição entre os operadores  $\mu$ ,  $\rho$  e  $\sigma$ . O operador de transição espacial,  $\mu$ , movimenta o agente de sua posição considerando-se os atributos de conectividade e mobilidade; o operador  $\rho$  realiza os contatos entre os agentes, que ocorrem com certa probabilidade; e o operador de transição temporal,  $\sigma$ , realiza a transição do estado interno do agente considerando sua interação com outros agentes e o ambiente. O operador de evolução  $\lambda$  é definido como em (3.2).

$$\lambda(\chi(t)^{(x,y)}) \equiv \sigma\left(\rho(\mu(\chi(t+1)))^{(\xi,\eta)}\right) \quad (3.2)$$

indicando que o agente  $\chi$  realizou uma evolução da posição  $(x, y)$  do ciclo  $t$  à posição  $(\xi, \eta)$  do ciclo posterior,  $t + 1$ . Assim sendo, o operador espaço-temporal  $\lambda(\chi)$  realiza as operações de movimentação, contato e de transição do agente  $\chi$  movimentando-o da posição  $(x, y)$  para uma posição  $(\xi, \eta)$  do tempo atual,  $t$ , para tempo posterior,  $t + 1$ .

### 3.4 Modelagem em *Bitstring* dos Agentes

A modelagem *bitstring* realizada para a representação do agente é baseada na manipulação direta dos *bits* em uma palavra computacional, que é capaz de caracterizar sem ambiguidade a especificação do agente  $\chi(t) = (L, X, Y, C, Q)$ , com identificador do lote,  $L$ , coordenada  $x$  da posição,  $X$ , coordenada  $y$  da posição,  $Y$ , contador de controle,  $C$ , e estado,  $Q$ .

Ao emprego de um modelo em *bitstring* é necessário utilizar uma linguagem de programação que dê suporte apropriado às operações diretas com *bits*. À implementação do sistema multiagente, visando alcançar os objetivos definidos neste trabalho, é proposta a utilização da linguagem de programação C, que provê suporte aos propósitos de modelagem e paralelização do sistema. O tipo de dado inteiro em C++, *int*, contém 32 *bits*, sendo suficiente à especificação da formulação para o agente  $\chi(t)$  em termos de quantidade de *bits*, como realizado a seguir.

À identificação do lote atual,  $L$ , do agente, considera-se que o identificador não ultrapasse 64 valores, de modo que se supõe que uma quadra contenha no máximo 64 lotes. À modelagem

da posição  $X, Y$  de um agente, considera-se que as quantidades de linhas,  $\#L$ , e de colunas,  $\#C$ , do ambiente, são limitadas por  $\max(\#L \times \#C) = (512 \times 512)$ , em que  $\max$  indica o máximo possível de linhas e de colunas que podem ser alocadas à essa escolha, considerando-se que uma aproximação matricial de um lote de uma quadra resulta em uma matriz de no máximo  $512 \times 512$  posições. À modelagem do contador de controle,  $C$ , de um agente, considera-se que a quantidade de ciclos em que um agente fica no estado exposto, infectado ou recuperado, não ultrapassem 64 ciclos de simulação, pois supõe-se que a doença hipotética modelada não tem períodos de transição superior à esse valor. À modelagem do estado  $Q$  de um agente, considerando a adoção do modelo compartmental tipo SEIRS, existem somente quatro distintos estados.

Assim sendo, observe que  $512_{10} = 2^9$ ,  $64_{10} = 2^6$  e  $4_{10} = 2^2$ , sendo suficiente uma palavra que comporte um quantidade de 32 bits para armazenar a especificação do agente  $\chi(t) = (L, X, Y, C, Q)$  em bits. A utilização de faixas maiores à representação dos atributos de um agente demandaria outros tipos de dados em C++, com maior quantidade de bits. A representação da palavra computacional proposta à modelagem dos agentes é como ilustrado na Figura 3.4, sendo que os elementos  $l_i \in L$ ,  $x_i \in X$ ,  $y_i \in Y$ ,  $c_i \in C$  e  $q_i \in Q$  possuem o valor 0 ou 1 e seus índices subscritos indicam que cada bit pode variar nos respectivos intervalos que estão especificados.

L	X	Y	C	Q
31   30   29   28   27   26   25   24   23   22   21   20   19   18   17   16   15   14   13   12   11   10   9   8   7   6   5   4   3   2   1   0				

Figura 3.4: Representação *bitstring* do agente  $\chi$ .

Na Figura 3.4, os elementos do identificador do lote  $L$  estão nas posições 31 a 26. Os elementos das coordenadas  $X$  e  $Y$  da posição do agente no lote estão nas posições 25 a 17 e 16 a 8, respectivamente. Os elementos do contador de controle,  $C$ , e do estado,  $Q$ , estão nas posições 7 a 2 e 1 a 0, respectivamente. Considerando as escolhas realizadas à sua modelagem, um agente qualquer do modelo pode ser representado em bits como em (3.3).

$$\chi(L, X, Y, C, Q) \equiv (l_{31}, \dots, l_{26}; x_{25}, \dots, x_{17}; y_{16}, \dots, y_8; c_7, \dots, c_2; q_1, q_0) \quad (3.3)$$

Para identificar o lote e as coordenadas da posição do agente  $\chi(L, X, Y, C, Q) = \chi(t)$  no ambiente, assim como seu estado interno e o registro de controle de ciclos às transições de

estados, essas características são definidas como em (3.4), para o ciclo  $t$ , em que  $Pl$ ,  $Px$ ,  $Py$ ,  $Pc$  e  $Pq$  designam, respectivamente, as quantidades de *bits* posteriores à cada campo.

$$\begin{cases} L(t) = \chi(L, X, Y, C, Q) \ll_{arth} (Pl) \gg_{arth} (C\#l) \\ X(t) = \chi(L, X, Y, C, Q) \ll_{arth} (Px) \gg_{arth} (C\#x) \\ Y(t) = \chi(L, X, Y, C, Q) \ll_{arth} (Py) \gg_{arth} (C\#y) \\ C(t) = \chi(L, X, Y, C, Q) \ll_{arth} (Pc) \gg_{arth} (C\#c) \\ Q(t) = \chi(L, X, Y, C, Q) \ll_{arth} (Pq) \gg_{arth} (C\#q) \end{cases} \quad (3.4)$$

A quantidade de *bits* posteriores à cada campo é a quantidade de *bits* existentes que são mais significativos que aqueles do campo em questão. Os termos  $C\#l$ ,  $C\#x$ ,  $C\#y$ ,  $C\#c$  e  $C\#q$  designam, respectivamente, as quantidades de *bits* complementares a cada campo. A quantidade de *bits* complementares a cada campo é a diferença entre a quantidade de *bits* total da palavra e a quantidade de *bits* do campo. As operações  $\ll_{arth}$  e  $\gg_{arth}$  designam, respectivamente, deslocamentos não circulares à esquerda e à direita na palavra, na quantidade de *bits* indicado. As notações  $\ll_{arth}$  e  $\gg_{arth}$  advém daquelas nomenclaturas utilizadas na linguagem C++ para designar as operações de deslocamento ou *shift* como deslocamentos aritméticos.

Em deslocamentos aritméticos, como aqueles que são definidos na linguagem C++, ocorre a preservação do *bit* de sinal, quando da sua execução no sentido esquerda-direita. Deslocamentos que não preservam o *bit* de sinal são chamados deslocamentos lógicos, que não são definidos na linguagem C++. O uso do tipo de dado *unsigned int* descarta a preservação de sinal na realização de *shifts* ou deslocamentos, evitando a introdução de erro na manipulação dos *bits* mais significativos em operações de captura e configuração de atributos de um agente, viabilizando o uso de deslocamentos aritméticos com o mesmo resultado prático obtido por deslocamentos lógicos, sendo o uso do tipo *unsigned int* em C++ justificado por este motivo.

Na modelagem realizada, as especificações às quantidades são  $Pl = 0$ ,  $Px = 6$ ,  $Py = 15$ ,  $Pc = 24$ ,  $Pq = 30$ ,  $C\#l = 26$ ,  $C\#x = 23$ ,  $C\#y = 23$ ,  $C\#c = 26$  e  $C\#q = 30$ , de modo que (3.4) é então reescrita concretamente para a modelagem (3.3) como em (3.5).

$$\begin{cases} L(t) = \chi(L, X, Y, C, Q) \ll_{arth} (0) \gg_{arth} (26) \\ X(t) = \chi(L, X, Y, C, Q) \ll_{arth} (6) \gg_{arth} (23) \\ Y(t) = \chi(L, X, Y, C, Q) \ll_{arth} (15) \gg_{arth} (23) \\ C(t) = \chi(L, X, Y, C, Q) \ll_{arth} (24) \gg_{arth} (26) \\ Q(t) = \chi(L, X, Y, C, Q) \ll_{arth} (30) \gg_{arth} (30) \end{cases} \quad (3.5)$$

As operações definidas em (3.5) capturam informações armazenadas nos agentes, como especificado em (3.1).

Assim sendo, para o modelo *bitstring* especificado neste trabalho, considera-se um agente  $\chi(t)$  especificado como em (3.6), em que o símbolo | designa o separador dos campos do agente.

$$\chi(t) = 110100|001010100|011001011|001110|10 \quad (3.6)$$

A aplicação do operador de captura do campo de identificador do lote em que o agente  $\chi(t)$  se encontra, como definido em (3.5), é exemplificada como ilustrado em (3.7).

$$\begin{aligned} L(t) &= \chi(t) \ll_{arth} (0) \gg_{arth} (26) \\ &= 110100|001010100|011001011|001110|10 \ll_{arth} (0) \gg_{arth} (26) \\ &= 110100|001010100|011001011|001110|10 \gg_{arth} (26) \quad (3.7) \\ &= 000000|000000000|000000000|001101|00 \\ &= 110100 \end{aligned}$$

A aplicação do operador de captura do campo da coordenada  $x$  do lote em que o agente  $\chi(t)$  se encontra, como definido em (3.5), é exemplificada como ilustrado em (3.8).

$$\begin{aligned} X(t) &= \chi(t) \ll_{arth} (6) \gg_{arth} (23) \\ &= 110100|001010100|011001011|001110|10 \ll_{arth} (6) \gg_{arth} (23) \\ &= 001010|100011001|011001110|100000|00 \gg_{arth} (23) \quad (3.8) \\ &= 000000|000000000|000000000|010101|00 \\ &= 001010100 \end{aligned}$$

A aplicação do operador de captura do campo da coordenada  $y$  do lote em que o agente  $\chi(t)$  se encontra, como definido em (3.5), é exemplificada como ilustrado em (3.9).

$$\begin{aligned} Y(t) &= \chi(t) \ll_{arth} (15) \gg_{arth} (23) \\ &= 110100|001010100|011001011|001110|10 \ll_{arth} (15) \gg_{arth} (23) \\ &= 011001|011001110|100000000|000000|00 \gg_{arth} (23) \quad (3.9) \\ &= 000000|000000000|000000000|110010|11 \\ &= 011001011 \end{aligned}$$

A aplicação do operador de captura do campo do contador de transição de estados do agente  $\chi(t)$ , como definido em (3.5), é exemplificada como ilustrado em (3.10).

$$\begin{aligned}
 C(t) &= \chi(t) \ll_{arth} (24) \gg_{arth} (26) \\
 &= 110100|001010100|011001011|001110|10 \ll_{arth} (24) \gg_{arth} (26) \\
 &= 001110|100000000|000000000|000000|00 \gg_{arth} (26) \\
 &= 000000|000000000|000000000|000011|10 \\
 &= 001110
 \end{aligned} \tag{3.10}$$

A aplicação do operador de captura do campo do estado do agente  $\chi(t)$ , como definido em (3.5), é exemplificada como ilustrado em (3.11).

$$\begin{aligned}
 Q(t) &= \chi(t) \ll_{arth} (30) \gg_{arth} (30) \\
 &= 110100|001010100|011001011|001110|10 \ll_{arth} (30) \gg_{arth} (30) \\
 &= 100000|000000000|000000000|000000|00 \gg_{arth} (30) \\
 &= 000000|000000000|000000000|000000|10 \\
 &= 10
 \end{aligned} \tag{3.11}$$

Também são necessárias outras operações para manipular os atributos dos agentes, além das definidas em (3.5), como, por exemplo, operadores de deslocamentos circulares. Como a linguagem de programação C++ não conta com operador próprio que viabilize a execução de operações de deslocamentos circulares, faz-se necessário a sua definição com base nos operadores existentes na linguagem, como os deslocamentos aritméticos e operações lógicas. Em (3.12)  $\ll_{circ}$  e  $\gg_{circ}$  designam, respectivamente, deslocamentos circulares à esquerda e à direita na palavra, na quantidade de *bits* indicado, definidos como uma composição dos operadores de deslocamento aritmético e a operação lógica ou.

$$\begin{cases} \ll_{circ} = (M \ll_{arth} (shift)) \vee (M \gg_{arth} (N\_BITS - shift)) \\ \gg_{circ} = (M \gg_{arth} (shift)) \vee (M \ll_{arth} (N\_BITS - shift)) \end{cases} \tag{3.12}$$

Em (3.12),  $M$  denota a palavra computacional em que está sendo aplicada a operação de deslocamento circular,  $N\_BITS$  denota a quantidade de bits totais,  $shift$  denota o fator de deslocamento e  $\vee$  denota a operação lógica OR, "ou inclusivo", que é realizada *bit a bit* à

palavra. Assim, a especificação às quantidades adotadas na modelagem, (3.12) é reescrita como em (3.13).

$$\begin{cases} \ll_{circ} = (M \ll_{arth} (shift)) \vee (M \gg_{arth} (32 - shift)) \\ \gg_{circ} = (M \gg_{arth} (shift)) \vee (M \ll_{arth} (32 - shift)) \end{cases} \quad (3.13)$$

Em (3.14) é ilustrado um exemplo da aplicação do operador  $\ll_{circ}$  definido em (3.13) em uma palavra computacional  $M = 11010000|10101000|11001011|00111010$  de 32 bits, com  $shift = 8$ .

$$\begin{aligned} \ll_{circ} &= (M \ll_{arth} (shift)) \vee (M \gg_{arth} (32 - shift)) \\ &= 11010000|10101000|11001011|00111010 \ll_{arth} (8) \vee \\ &\quad 11010000|10101000|11001011|00111010 \gg_{arth} (32 - 8) \\ &= 11010000|10101000|11001011|00111010 \ll_{arth} (8) \vee \\ &\quad 11010000|10101000|11001011|00111010 \gg_{arth} (24) \\ &= 10101000|11001011|00111010|00000000 \vee \\ &\quad 00000000|00000000|00000000|11010000 \\ &= 10101000|11001011|00111010|11010000 \end{aligned} \quad (3.14)$$

Em (3.15) é ilustrado um exemplo da aplicação do operador  $\gg_{circ}$  definido em (3.13) em uma palavra computacional  $M = 11010000|10101000|11001011|00111010$  de 32 bits, com  $shift = 8$ .

$$\begin{aligned}
\gg_{circ} &= (M \gg_{arth} (shift)) \vee (M \ll_{arth} (32 - shift)) \\
&= 11010000|10101000|11001011|00111010 \gg_{arth} (8) \vee \\
&\quad 11010000|10101000|11001011|00111010 \ll_{arth} (32 - 8) \\
&= 11010000|10101000|11001011|00111010 \gg_{arth} (8) \vee \\
&\quad 11010000|10101000|11001011|00111010 \ll_{arth} (24) \tag{3.15} \\
&= 00000000|11010000|10101000|11001011 \vee \\
&\quad 00111010|00000000|00000000|00000000 \\
&= 00111010|11010000|10101000|11001011
\end{aligned}$$

Subsequentemente são utilizadas as operações definidas em (3.13), para implementar o operador espaço-temporal  $\lambda(\chi(t))$  que movimenta o agente da posição  $(x, y)$  para uma posição  $(\xi, \eta)$  no ciclo de tempo atual  $t$ , para o ciclo de tempo  $t + 1$ . As operações definidas em (3.16) resultam na atualização dos atributos de identificação do lote, das coordenadas  $x$  e  $y$  da posição, do controle e do estado do agente.

$$\left\{
\begin{array}{l}
\chi(L(t+1), X(t), Y(t), C(t), Q(t)) = \begin{aligned} &\chi(L(t), X(t), Y(t), C(t), Q(t)) \\ &\gg_{circ} (Al) \gg_{arth} (\#l) \ll_{arth} (\#l) \\ &\vee L(t+1) \ll_{circ} (Al) \end{aligned} \\
\chi(L(t), X(t+1), Y(t), C(t), Q(t)) = \begin{aligned} &\chi(L(t), X(t), Y(t), C(t), Q(t)) \\ &\gg_{circ} (Ax) \gg_{arth} (\#x) \ll_{arth} (\#x) \\ &\vee X(t+1) \ll_{circ} (Ax) \end{aligned} \\
\chi(L(t), X(t), Y(t+1), C(t), Q(t)) = \begin{aligned} &\chi(L(t), X(t), Y(t), C(t), Q(t)) \\ &\gg_{circ} (Ay) \gg_{arth} (\#y) \ll_{arth} (\#y) \\ &\vee Y(t+1) \ll_{circ} (Ay) \end{aligned} \tag{3.16} \\
\chi(L(t), X(t), Y(t), C(t+1), Q(t)) = \begin{aligned} &\chi(L(t), X(t), Y(t), C(t), Q(t)) \\ &\gg_{circ} (Ac) \gg_{arth} (\#c) \ll_{arth} (\#c) \\ &\vee C(t+1) \ll_{circ} (Ac) \end{aligned} \\
\chi(L(t), X(t), Y(t), C(t), Q(t+1)) = \begin{aligned} &\chi(L(t), X(t), Y(t), C(t), Q(t)) \\ &\gg_{circ} (Aq) \gg_{arth} (\#q) \ll_{arth} (\#q) \\ &\vee Q(t+1) \ll_{circ} (Aq) \end{aligned}
\end{array}
\right.$$

Em (3.16),  $Al$ ,  $Ax$ ,  $Ay$ ,  $Ac$  e  $Aq$  designam, respectivamente, as quantidades de *bits* anteriores a cada campo. A quantidade de *bits* anteriores a cada campo é a quantidade de *bits* existentes

que são menos significativos que aqueles do campo em questão. Os termos  $\#l$ ,  $\#x$ ,  $\#y$ ,  $\#c$  e  $\#q$  designam, respectivamente, as quantidades de *bits* de cada campo. Como  $Al = 26$ ,  $Ax = 17$ ,  $Ay = 8$ ,  $Ac = 2$ ,  $Aq = 0$ ,  $\#l = 6$ ,  $\#x = 9$ ,  $\#y = 9$ ,  $\#c = 6$  e  $\#q = 2$ , a especificação às quantidades adotadas na modelagem, (3.16) é reescrita como em (3.17).

$$\left\{ \begin{array}{ll} \chi(L(t+1), X(t), Y(t), C(t), Q(t)) = & \chi(L(t), X(t), Y(t), C(t), Q(t)) \\ & \gg_{circ} (26) \gg_{arth} (6) \ll_{arth} (6) \\ & \vee L(t+1) \ll_{circ} (26) \\ \chi(L(t), X(t+1), Y(t), C(t), Q(t)) = & \chi(L(t), X(t), Y(t), C(t), Q(t)) \\ & \gg_{circ} (17) \gg_{arth} (9) \ll_{arth} (9) \\ & \vee X(t+1) \ll_{circ} (17) \\ \chi(L(t), X(t), Y(t+1), C(t), Q(t)) = & \chi(L(t), X(t), Y(t), C(t), Q(t)) \\ & \gg_{circ} (8) \gg_{arth} (9) \ll_{arth} (9) \\ & \vee Y(t+1) \ll_{circ} (8) \\ \chi(L(t), X(t), Y(t), C(t+1), Q(t)) = & \chi(L(t), X(t), Y(t), C(t), Q(t)) \\ & \gg_{circ} (2) \gg_{arth} (6) \ll_{arth} (6) \\ & \vee C(t+1) \ll_{circ} (2) \\ \chi(L(t), X(t), Y(t), C(t), Q(t+1)) = & \chi(L(t), X(t), Y(t), C(t), Q(t)) \\ & \gg_{circ} (0) \gg_{arth} (2) \ll_{arth} (2) \\ & \vee Q(t+1) \ll_{circ} (0) \end{array} \right. \quad (3.17)$$

Considere novamente o agente  $\chi(t)$  especificado em (3.6), em que o símbolo  $|$  designa o separador dos campos do agente. A aplicação do operador definido em (3.17) para a configuração do campo do identificador do lote do agente  $\chi(t)$  de  $L(t)$  para  $L(t+1)$ , é exemplificada como ilustrado em (3.18), considerando  $L(t+1) = 001111$ .

$$\begin{aligned}
\chi(t+1) = & \chi(t) \gg_{circ} (26) \gg_{arth} (6) \ll_{arth} (6) \vee L(t+1) \ll_{circ} (26) \\
= & 110100|001010100|011001011|001110|10 \gg_{circ} (26) \\
& \gg_{arth} (6) \ll_{arth} (6) \vee 001111 \ll_{circ} (26) \\
= & 001010|100011001|011001110|101101|00 \gg_{arth} (6) \\
& \ll_{arth} (6) \vee 001111 \ll_{circ} (26) \\
= & 000000|001010100|011001011|001110|10 \ll_{arth} (6) \\
& \vee 001111 \ll_{circ} (26) \\
= & 001010|100011001|011001110|100000|00 \vee 001111 \ll_{circ} (26) \\
= & 001010|100011001|011001110|100011|11 \ll_{circ} (26) \\
= & 001111|001010100|011001011|001110|10
\end{aligned} \tag{3.18}$$

A aplicação do operador definido em (3.17) para a configuração do campo da coordenada  $x$  do lote do agente  $\chi(t)$  de  $X(t)$  para  $X(t+1)$ , é exemplificada como ilustrado em (3.19), considerando  $X(t+1) = 110000111$ .

$$\begin{aligned}
\chi(t+1) = & \chi(t) \gg_{circ} (17) \gg_{arth} (9) \ll_{arth} (9) \vee X(t+1) \ll_{circ} (17) \\
= & 110100|001010100|011001011|001110|10 \gg_{circ} (17) \\
& \gg_{arth} (9) \ll_{arth} (9) \vee 110000111 \ll_{circ} (17) \\
= & 011001|011001110|101101000|010101|00 \gg_{arth} (9) \\
& \ll_{arth} (9) \vee 110000111 \ll_{circ} (17) \\
= & 000000|000011001|011001110|101101|00 \ll_{arth} (9) \\
& \vee 110000111 \ll_{circ} (17) \\
= & 011001|011001110|101101000|000000|00 \vee 110000111 \ll_{circ} (17) \\
= & 011001|011001110|101101001|100001|11 \ll_{circ} (17) \\
= & 110100|110000111|011001011|001110|10
\end{aligned} \tag{3.19}$$

A aplicação do operador definido em (3.17) para a configuração do campo da coordenada  $y$  do lote do agente  $\chi(t)$  de  $Y(t)$  para  $Y(t + 1)$ , é exemplificada como ilustrado em (3.20), considerando  $Y(t + 1) = 111101000$ .

$$\begin{aligned}
\chi(t + 1) &= \chi(t) \gg_{circ} (8) \gg_{arth} (9) \ll_{arth} (9) \vee Y(t + 1) \ll_{circ} (8) \\
&= 110100|001010100|011001011|001110|10 \gg_{circ} (8) \\
&\quad \gg_{arth} (9) \ll_{arth} (9) \vee 111101000 \ll_{circ} (8) \\
&= 001110|101101000|010101000|110010|11 \gg_{arth} (9) \\
&\quad \ll_{arth} (9) \vee 111101000 \ll_{circ} (8) \\
&= 000000|000001110|101101000|010101|00 \ll_{arth} (9) \\
&\quad \vee 111101000 \ll_{circ} (8) \\
&= 001110|101101000|010101000|000000|00 \vee 111101000 \ll_{circ} (8) \\
&= 001110|101101000|010101001|111010|00 \ll_{circ} (8) \\
&= 110100|001010100|111101000|001110|10
\end{aligned} \tag{3.20}$$

A aplicação do operador definido em (3.17) para a configuração do campo do contador de transição de estados do agente  $\chi(t)$  de  $C(t)$  para  $C(t + 1)$ , é exemplificada como ilustrado em (3.21), considerando  $C(t + 1) = 101011$ .

$$\begin{aligned}
\chi(t+1) &= \chi(t) \gg_{circ} (2) \gg_{arth} (6) \ll_{arth} (6) \vee C(t+1) \ll_{circ} (2) \\
&= 110100|001010100|011001011|001110|10 \gg_{circ} (2) \\
&\quad \gg_{arth} (6) \ll_{arth} (6) \vee 101011 \ll_{circ} (2) \\
&= 101101|000010101|000110010|110011|10 \gg_{arth} (6) \\
&\quad \ll_{arth} (6) \vee 101011 \ll_{circ} (2) \\
&= 000000|101101000|010101000|110010|11 \ll_{arth} (6) \\
&\quad \vee 101011 \ll_{circ} (2) \\
&= 101101|000010101|000110010|110000|00 \vee 101011 \ll_{circ} (2) \\
&= 101101|000010101|000110010|111010|11 \ll_{circ} (2) \\
&= 110100|001010100|011001011|101011|10
\end{aligned} \tag{3.21}$$

A aplicação do operador definido em (3.17) para a configuração do campo do estado do agente  $\chi(t)$  de  $Q(t)$  para  $Q(t+1)$ , é exemplificada como ilustrado em (3.22), considerando  $Q(t+1) = 01$ .

$$\begin{aligned}
\chi(t+1) &= \chi(t) \gg_{circ} (0) \gg_{arth} (2) \ll_{arth} (2) \vee Q(t+1) \ll_{circ} (0) \\
&= 110100|001010100|011001011|001110|10 \gg_{circ} (0) \\
&\quad \gg_{arth} (2) \ll_{arth} (2) \vee 01 \ll_{circ} (0) \\
&= 110100|001010100|011001011|001110|10 \gg_{arth} (2) \\
&\quad \ll_{arth} (2) \vee 01 \ll_{circ} (0) \\
&= 001101|000010101|000110010|110011|10 \ll_{arth} (2) \\
&\quad \vee 01 \ll_{circ} (0) \\
&= 110100|001010100|011001011|001110|00 \vee 01 \ll_{circ} (0) \\
&= 110100|001010100|011001011|001110|01 \ll_{circ} (0) \\
&= 110100|001010100|011001011|001110|01
\end{aligned} \tag{3.22}$$

Na Tabela 3.1 são apresentados sinteticamente os valores relativos às quantidades de *bits* dos tamanhos, complementos, posteriores e anteriores de cada atributo do agente  $\chi$ .

Atributo	Tamanho	Complementar	Posteriores	Anteriores
Identificador do Lote $L$	6	26	0	26
Coordenada $X$	9	23	6	17
Coordenada $Y$	9	23	15	8
Contador de Controle $C$	6	26	24	2
Estado $Q$	2	30	30	0

Tabela 3.1: Tabela das variáveis relacionadas aos campos dos agentes.

Para completar a modelagem é suficiente detalhar e relacionar os identificadores dos lotes e dimensões dos lotes, a faixa de variação do controle, e os possíveis tipos de estados do agente na base 10, com tais características na base 2, para determinar sem ambiguidade os elementos de (3.1).

Os identificadores dos lotes têm variação de  $(0)_{10} = (000\ 000)_2$  até  $(63)_{10} = (111\ 111)_2$ , totalizando 64 possíveis valores. Os lotes têm dimensão máxima de  $512_{10} = 2^9$  linhas por  $512_{10} = 2^9$  colunas, sendo possível variarem numa representação matricial da posição  $(0; 0)_{10} = (000\ 000\ 000; 000\ 000\ 000)_2$  até a posição  $(511; 511)_{10} = (111\ 111\ 111; 111\ 111\ 111)_2$ , totalizando os 512 possíveis valores. Semelhantemente, faz-se uma representação matricial ao controle para que sua variação ocorra de  $(0)_{10} = (000\ 000)_2$  até  $(63)_{10} = (111\ 111)_2$ , totalizando os 64 possíveis valores. Os estados do agente são setados como  $(0, 0)_2$  para o suscetível,  $S$ ,  $(0, 1)_2$  para o exposto,  $E$ ,  $(1, 0)_2$  para o infectante,  $I$ , e  $(1, 1)_2$  para o recuperado,  $R$ .

## 3.5 SIMULA

Como suporte às operações específicas relacionadas à manipulação de dados georreferenciados relativos às quadras utilizou-se um *software* especificamente desenvolvido para tal função, denominado SIMULA. Ele desempenha as funções de aquisição, tratamento e disponibilização de informações georreferenciadas para o programa que executa as simulações e para um módulo visualizador de saídas, sendo um *software* que integra operações de pré-processamento, como aquisição e tratamento de dados e de pós-processamento, como a visualização de saídas gráficas de arquivos resultantes das simulações executadas.

Na etapa de aquisição dos dados, o *software* realiza uma consulta em um Sistema Gerenciador de Banco de Dados Objeto Relacional, SGBDOR, utilizando a linguagem de consulta *Structured Query Language*, SQL [SQL 2016]. Nesta etapa são obtidas informações sobre os pontos georreferenciados que estão nos vértices de cada lote das quadras. As informações sobre os vértices são importantes pois definem a geometria do lote, que é a estrutura geométrica elementar à especificação das soluções, sendo utilizados posteriormente, após processamentos, à construção das estruturas matriciais que representam os lotes na simulação. Foi utilizado o SGBDOR PostgreSQL [PostgreSQL 2016] com a adição da extensão PostGIS [PostGIS 2016], que viabiliza o armazenamento e processamento de objetos com informações georreferenciadas em bancos de dados. Por meio da extensão PostGIS, os dados são importados para o banco de dados por meio de um arquivo em formato *shapefile* [Shapefile 2016], que é um formato de arquivos utilizado para o armazenamento de dados geoespaciais. O arquivo *shapefile* foi obtido em parceria com a prefeitura da cidade de Cascavel/PR.

Na etapa posterior à aquisição ocorre o tratamento dos dados obtidos, que é necessário para a remoção de informações redundantes que vem do banco de dados, e a adição de informações pertinentes que viabilizem, essencialmente, a aproximação da geometria dos lotes em matrizes bidimensionais. As principais processos executados dentro da etapa de tratamento de dados são:

- **Limpeza de pontos:** São removidos os pontos duplicados ou pontos que estejam muito próximos uns dos outros, utilizando como critério de remoção os erros relativos entre as latitudes e longitudes dos pontos.
- **Inserção de pontos:** É inserido um ponto adicional ao conjunto de pontos de um lote que contenha exatamente 6 pontos. Essa inserção de um ponto adicional é essencial para a realização correta da etapa de divisão de lotes.
- **Ordenação de pontos:** Os pontos são ordenados, em sentido horário, com início no ponto com menor latitude e maior longitude. A ordenação dos pontos é importante para a determinação do local apropriado à divisão de um lote.
- **Divisão de lotes:** É realizada a divisão de lotes que contenham 7 pontos, em dois outros sub-lotes. A divisão é executada por meio do cruzamento das diversas retas formadas

por dois pontos quaisquer do conjunto original de pontos, com o intuito de formar dois polígonos regulares, que gerarão os dois novos lotes.

- **Cálculo das dimensões dos lotes:** São calculados o comprimento e largura dos lotes, a partir das distâncias euclidianas dos pontos que estão nos vértices dos lotes.
- **Interpolação de pontos:** Por meio de interpolação linear, são interpolados os pontos entre os vértices, que formam as arestas dos lotes. Por meio dos pontos das arestas dos lotes, são interpolados linearmente os pontos internos ao lote. A interpolação é um processo importante no tratamento dos dados pois viabiliza a aproximação de uma malha que represente a geometria do lote, por meio de seus vértices.
- **Média de pontos:** São calculadas as médias entre subconjuntos de 4 pontos do conjunto de pontos obtidos da malha gerada pela etapa de interpolação de pontos. O processo realizado assemelha-se a uma convolução com máscaras, utilizando uma máscara de dimensão  $2 \times 2$ , com todos os coeficientes iguais a 1, e uma constante de normalização igual a  $\frac{1}{4}$ . Como resultado, obtém-se um conjunto de pontos em que, cada ponto, corresponde ao centro de uma posição na matriz do lote.
- **Cálculo das dimensões das matrizes:** Por meio dos pontos obtidos na etapa anterior são calculadas as dimensões das matrizes que representarão os lotes na simulação. Efetivamente, são calculadas as quantidades de linhas e de colunas das matrizes.

Durante o tratamento dos dados ocorre ainda o cálculo das posições vizinhas entre as fronteiras de cada lote. Durante este processo, para cada lote, calculam-se as posições vizinhas em outro lote para cada posição que está em sua fronteira, por meio de algoritmos especialmente desenvolvidos, o que possibilita que, os indivíduos, durante a operação de movimentação, desloquem-se de um lote para outro, se estiverem em uma posição de fronteira que possua posições vizinhas em um lote adjacente. Neste processo, os algoritmos necessitam apenas das quantidades de linhas e colunas de cada lote e as posições relativas entre os lotes. As informações sobre as posições relativas entre os lotes são lidas pelo SIMULA de um arquivo de configuração à parte, que foi gerado manualmente. Neste arquivo constam efetivamente as vizinhanças entre os lotes, informando para cada lote, quais lotes estão em sua vizinhança e quais

são as posições relativas entre eles. Estes algoritmos utilizam as informações de vizinhanças entre lotes para determinar qual método utilizar no cálculo das vizinhanças entre posições dos lotes: método de linhas, método de colunas ou método de canto. O método de linha é utilizado em vizinhanças horizontais, em que os pontos centrais dos lotes têm longitudes aproximadas e latitudes distantes. O método de colunas é utilizado em vizinhanças verticais, em que os pontos centrais dos lotes têm latitudes aproximadas e longitudes distantes. O método de canto é utilizado em vizinhanças em que os pontos centrais dos lotes têm latitudes e longitudes distantes. Após determinado o método, o algoritmo percorre todas as posições de fronteira do lote, calculando suas posições vizinhas nos lotes vizinhos, utilizando as informações sobre a quantidade de linhas e de colunas das matrizes dos lotes. Após o cálculo das vizinhanças, um arquivo texto é gerado e disponibilizado para a sistema de simulação, que irá realizar a leitura dos dados e construir a estrutura de vizinhança que é utilizada durante as simulações.

Na operação de processamento são executadas efetivamente as simulações. Após a geração de diversos arquivos de configuração, o *software* SIMULA executa o sistema de simulação, que realiza a leitura dos arquivos gerados e executa as simulações solicitadas. Ao final de cada simulação, o sistema de simulação gera arquivos texto de saída, que são utilizados pelo SIMULA para exibição de saídas gráficas na etapa de pós-processamento. É importante notar que, o sistema de simulação é independente ao *software* SIMULA, podendo ser utilizado separadamente, sendo necessário a geração ou cópia manual dos arquivos de entrada e algum meio independente para geração de saídas gráficas. Efetivamente, quando na execução dos experimentos numérico-computacionais, o sistema de simulação é utilizado separadamente ao SIMULA, por questões de simplicidade na execução de uma grande quantidade de simulações com diferentes parâmetros.

Na operação de pós-processamento, os arquivos de saída gerados pela simulação são utilizados para a visualização gráfica dos resultados no SIMULA. A Figura 3.5 ilustra a saída espacial obtida por meio da utilização das informações georreferenciadas da quadra 445, em que os indivíduos em azul representam indivíduos em estado suscetível, em amarelo representam indivíduos em estado latente, em vermelho representam indivíduos em estado infectado e em rosa representam indivíduos em estado recuperado. Esta visualização georreferenciada foi concebida e desenvolvida com base em aprimoramentos dos trabalhos realizados em [Bortoluzzi 2016].



Figura 3.5: Ilustração da saída espacial na quadra 445 gerada pelo SIMULA

A Figura 3.6 ilustra outro tipo de saída gráfica gerada pelo *software* SIMULA, os gráficos de curvas, que representam as quantidades de cada tipo de indivíduo durante o tempo de simulação. A linha em verde indica a quantidade de indivíduos em estado suscetível, em amarelo indica a quantidade de indivíduos em estado latente, em vermelho indica a quantidade de indivíduos em estado infectado e em azul indica a quantidade de indivíduos em estado recuperado.

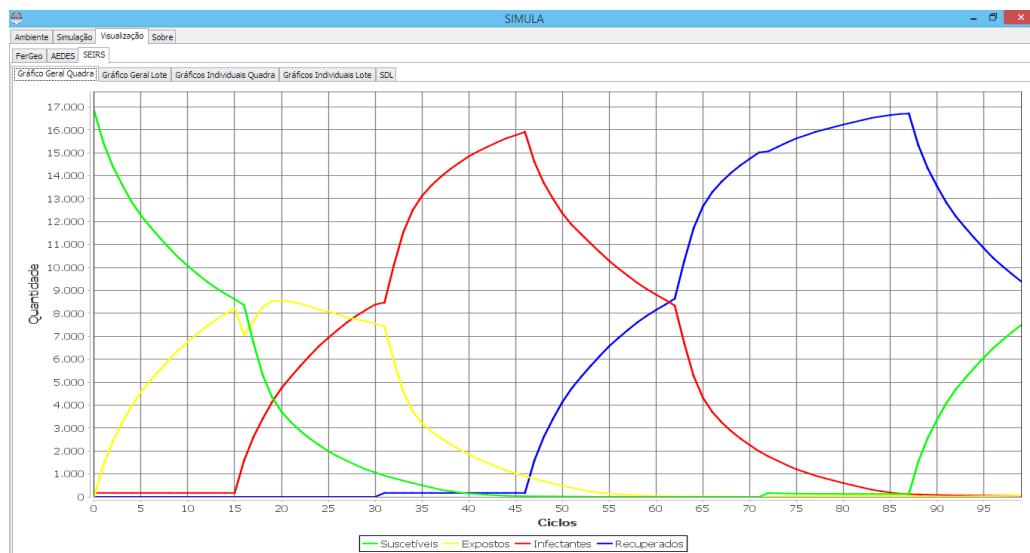


Figura 3.6: Ilustração do gráfico de linha gerado pelo SIMULA

A Figura 3.7 ilustra um terceiro tipo de saída gráfica gerada pelo *software* SIMULA, as saídas de espalhamento das populações em determinados lotes. Esta saída apresenta a distribuição

espacial das diferentes populações nos lotes de acordo com o tempo. Graficamente, é exibida uma matriz com dimensões idênticas à de determinado lote, em que cada posição é exibida de uma cor, dependendo dos estados dos indivíduos que a ocupam. Uma posição em cor verde indica que nesta posição existem somente indivíduos em estado suscetível, em cor amarela indica que existem indivíduos em estado latente e podem ou não existir indivíduos em estado suscetível, em cor vermelha indica que existem indivíduos em estado infectado e podem ou não existirem indivíduos em estado latente e suscetível, em cor azul indica que existem indivíduos em estado recuperado e podem ou não existirem indivíduos em estado infectado, latente e suscetível e em cor branca indica que não existem indivíduos ocupando tal posição. Cada ciclo da simulação gera uma matriz para cada lote, que é utilizado para compor esta saída.



Figura 3.7: Ilustração da saída espalhamento em um lote

Na Figura 3.8 são ilustradas as etapas descritas anteriormente e sua ordem de execução, contemplando o pré-processamento, processamento e pós-processamento.

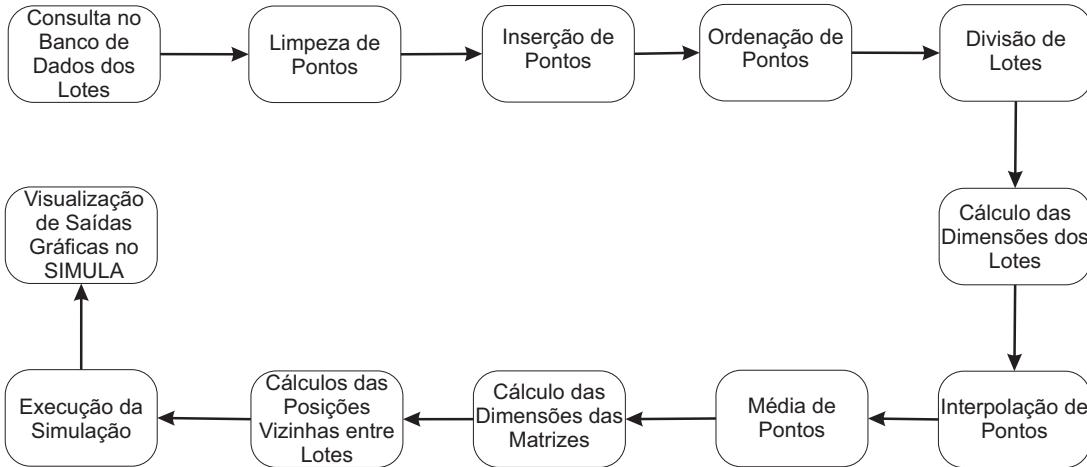


Figura 3.8: Fluxograma da execução completa do SIMULA.

### 3.6 Estruturas de Dados e Estratégias de Implementação

Para a implementação do sistema de simulação, mais efetivamente às operações de movimentação, contato e transição de estados dos indivíduos, utilizou-se estruturas de dados puramente vetoriais, ou seja, que são armazenadas em espaços de memória contíguos na memória do computador. A motivação do uso de estruturas vetoriais advém do fato de que reduzem a complexidade estrutural no código-fonte e viabilizam a posterior paralelização em CUDA por facilitar a cópia de dados entre a CPU e a GPU. Estruturas dinâmicas complexas, que suportam quantidades variáveis de elementos e garantem maior flexibilidade ao programador, são de difícil trato às operações de cópia de dados entre dispositivos, pois necessitam de métodos específicos para a serialização e a desserialização de dados, que acarretam em aumento na carga de processamento. A vetorização das estruturas de dados foi possível graças à simplicidade do indivíduo e do ambiente modelado na simulação, em que é possível expressar completamente seu significado por meio de vetores de tipo homogêneo e com quantidade fixa de elementos.

Quanto à estrutura matricial utilizada à representação dos lotes da quadra 445, todos os lotes representados na simulação armazenam, em cada posição de sua matriz, um vetor contendo os índices de todos os indivíduos que ocupam esta posição. Cada indivíduo tem um índice único, que corresponde à sua posição no vetor que armazena todos os indivíduos. Assim, embora as estruturas dos lotes são referidas como matrizes bidimensionais ou tridimensionais, efetivamente, na implementação realizada, estas estruturas são decompostas em vetores unidimensionais, sendo citadas como matrizes somente com o intuito de facilitar a compreensão em alto

nível da estrutura da quadra e de seus lotes, já que o mapeamento geográfico ao lógico assim foi realizado. Os estudos desenvolvidos durante a escrita de [Kaizer et al. 2016] serviram como base à definição da estrutura dos lotes da quadra 445, pois o armazenamento dos indivíduos que ocupam cada posição do lote na própria estrutura do lote, é um fator importante na otimização do processamento necessário nas operações de movimentação e de contato entre indivíduos. A Figura 3.9 ilustra um exemplo da estrutura aplicada a cada lote modelado neste trabalho, em que as hipotéticas posições do ambiente, mostradas nas cores verde e azul, armazenam vetores com os índices de todos os indivíduos que a ocupam. Os índices desses vetores são utilizados para acessar diretamente os indivíduos no vetor global de agentes.

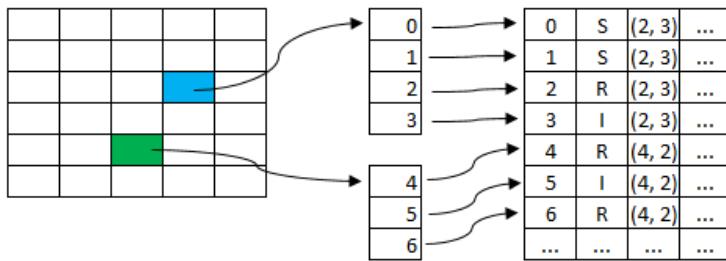


Figura 3.9: Ilustração da estrutura matricial empregada para a representação do mapeamento lógico dos lotes

As alterações que precisam ser realizadas, em termos de estruturas de dados, de uma implementação convencional à *bitstring* são mínimas. Em uma implementação convencional, utilizando o paradigma de programação orientada à objetos, utilizam-se classes para armazenar os atributos dos indivíduos e métodos que manipulam esses atributos. Assim, para armazenar todos os indivíduos em memória, é necessário utilizar um vetor de classes. Quando da aplicação da técnica de *bitstring*, substituem-se essas classes por um tipo de dado primitivo da linguagem de programação em uso.

Neste trabalho adotou-se a linguagem de programação C++ e seu tipo primitivo para números inteiros *int*. Assim, esse tipo primitivo é utilizado para armazenar efetivamente os atributos dos indivíduos e funções externas à estrutura do indivíduo são definidas para manipular seus atributos, decorrendo que, para o armazenamento de todos os indivíduos, é necessário apenas um vetor de inteiros ao invés de um vetor de classes. As funções externas necessárias para manipular os atributos dos indivíduos sobre o tipo de dado primitivo são aquelas definidas na

seção 3.4. Essa funções ou operações, quando na aplicação da técnica *bitstring*, equivalem-se a definição de métodos *get* e *set* na classe que representa o indivíduo, considerando-se o uso do paradigma de programação orientada à objetos.

Os algoritmos 1, 2 e 3 ilustram os pseudo-códigos em alto nível às operações de movimentação, contato e transição de estados dos indivíduos, que foram efetivamente implementados no sistema de simulação. É importante perceber que, os pseudo-códigos apresentados podem ser utilizados tanto para realizar uma implementação convencional, utilizando o paradigma de programação orientada à objetos, quanto para implementar uma versão utilizando a técnica de *bitstring*, pois, quando da definição dos métodos para manipulação dos atributos dos indivíduos, seja ela interna ou externamente à uma classe, os resultados produzidos por essas operações, são equivalentes.

---

**Algoritmo 1:** MOVIMENTAÇÃO DOS INDIVÍDUOS

---

```
1 início
2   para cada indivíduo na simulação faça
3     direção = randomiza_um_número_inteiro_entre(0, 8)
4     se direção = 0 então
5       | subtrai_um_coordenada_x(indivíduo)
6     fim
7     se direção = 1 então
8       | soma_um_coordenada_x(indivíduo)
9     fim
10    se direção = 2 então
11      | subtrai_um_coordenada_y(indivíduo)
12    fim
13    se direção = 3 então
14      | soma_um_coordenada_y(indivíduo)
15    fim
16    se direção = 4 então
17      | subtrai_um_coordenada_x(indivíduo)
18      | subtrai_um_coordenada_y(indivíduo)
19    fim
20    se direção = 5 então
21      | subtrai_um_coordenada_x(indivíduo)
22      | soma_um_coordenada_y(indivíduo)
23    fim
24    se direção = 6 então
25      | soma_um_coordenada_x(indivíduo)
26      | subtrai_um_coordenada_y(indivíduo)
27    fim
28    se direção = 7 então
29      | soma_um_coordenada_x(indivíduo)
30      | soma_um_coordenada_y(indivíduo)
31    fim
32    se direção = 8 então
33      | se posição(indivíduo) tem vizinhos em outro lote então
34        |   move_individuo_para_posição_vizinha_aleatória()
35      | fim
36    fim
37  fim
38 fim
```

---

---

**Algoritmo 2:** CONTATO ENTRE INDIVÍDUOS

---

```
1 início
2   para cada lote  $k$  da quadra faça
3     para cada posição  $(x, y)$  do lote  $k$  faça
4       se posição  $(x, y)$  contém indivíduos infectados então
5         para cada indivíduo suscetível na posição  $(x, y)$  faça
6           se percentual_randômico  $\leq \beta$  então
7             altera_estado_do_indivíduo_para_exposto()
8           fim
9         fim
10      fim
11    fim
12  fim
13 fim
```

---

---

**Algoritmo 3:** TRANSIÇÃO DE ESTADOS DOS INDIVÍDUOS

---

```
1 para cada indivíduo na simulação faça
2   se indivíduo está em estado exposto então
3     se contador de períodos do indivíduo  $\geq \gamma$  então
4       altera_estado_do_indivíduo_para_infectado()
5       zera_contador_de_períodos_do_indivíduo()
6     senão
7       soma_um_contador_de_períodos_do_indivíduo()
8     fim
9   fim
10  se indivíduo está em estado infectado então
11    se contador de períodos do indivíduo  $\geq \alpha$  então
12      altera_estado_do_indivíduo_para_recuperado()
13      zera_contador_de_períodos_do_indivíduo()
14    senão
15      soma_um_contador_de_períodos_do_indivíduo()
16    fim
17  fim
18  se indivíduo está em estado recuperado então
19    se contador de períodos do indivíduo  $\geq \delta$  então
20      altera_estado_do_indivíduo_para_suscetível()
21      zera_contador_de_períodos_do_indivíduo()
22    senão
23      soma_um_contador_de_períodos_do_indivíduo()
24    fim
25  fim
26 fim
```

---

As Figuras 3.10, 3.11 e 3.12 ilustram fluxogramas para as operações de movimentação, contato e transição de estados dos indivíduos, respectivamente.

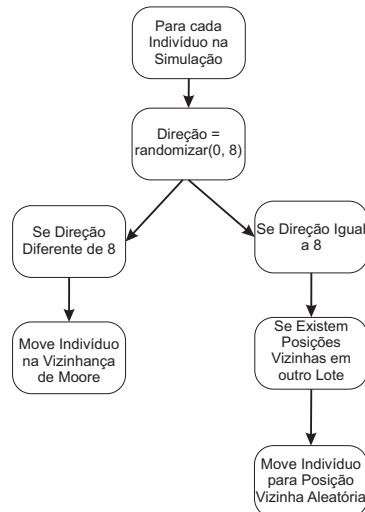


Figura 3.10: Fluxograma para a operação de movimentação dos indivíduos.

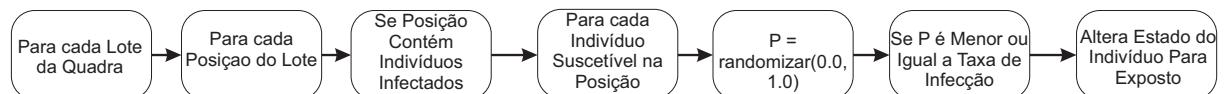


Figura 3.11: Fluxograma para a operação de contato entre os indivíduos.

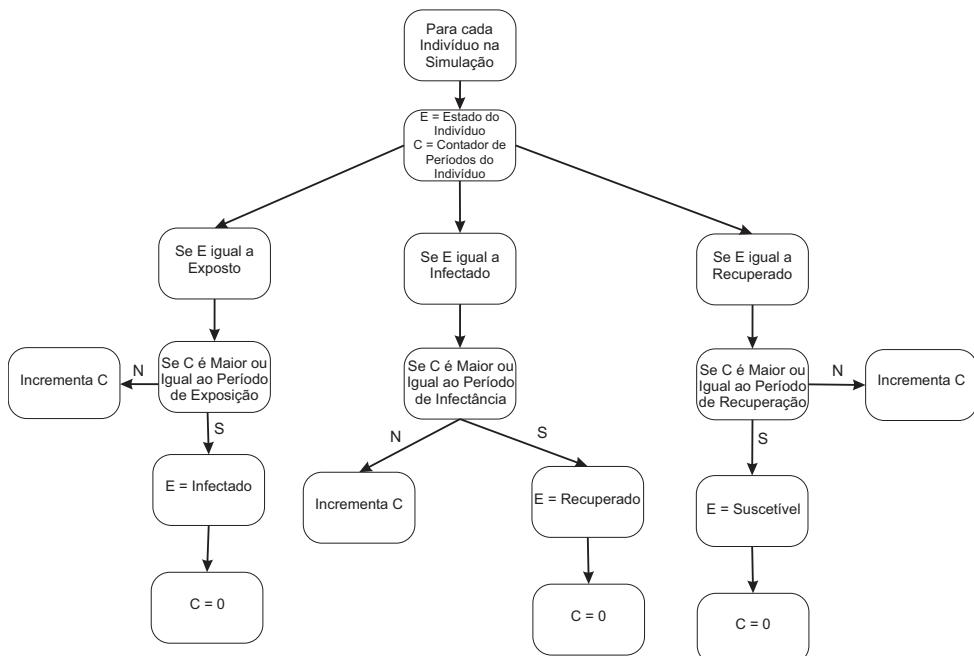


Figura 3.12: Fluxograma para a operação de transição de estados dos indivíduos.

# **Capítulo 4**

## **Soluções: Resultados e Discussões**

### **4.1 Introdução**

Neste capítulo são apresentados e discutidos os resultados obtidos por meio da execução de simulações utilizando as implementações realizadas. São apresentados os tempos de execução e os *speedups* obtidos com a execução dos testes, bem como os resultados visuais para a dinâmica, por meio das saídas espaço-temporais geradas com auxílio do *software* SIMULA e as curvas de quantidades, que expressam as quantidades de estado da população durante o tempo de simulação. São comparados empiricamente os resultados obtidos, tanto pela execução das simulações de forma sequencial e paralela, quanto pelas estratégias de implementação empregadas, como a convencional e a *bitstring*. Por fim são apresentadas as conclusões obtidas a partir dos estudos realizados durante o desenvolvimento deste trabalho e sugestões para atividades ou trabalhos futuros.

### **4.2 Discussões Qualitativas, Quantitativas, Eficiência e Acurácia**

Na Tabela 4.1 são apresentados os parâmetros utilizados na execução das simulações. À obtenção dos resultados foi empregada a metodologia de Monte Carlo [Chwif e Medina 2014], que consiste na execução de determinada quantidade de simulações, tomando-se ao final a média aritmética, que é utilizada para a análise dos resultados. A distribuição inicial das quantidades de indivíduos foi realizada de forma uniforme entre os lotes, ou seja, em cada lote são inseridos  $16830/17 = 990$  indivíduos no estado suscetível e  $170/17 = 10$  no estado infectado.

Parâmetro	Valor Mínimo	Valor Máximo
Quantidade de Simulações	100	100
Quantidade de Ciclos	500	500
Quantidade de Suscetíveis	16830	16830
Quantidade de Expostos	0	0
Quantidade de Infectados	170	170
Quantidade de Recuperados	0	0
Taxa de Infecção $\beta$	0.90	0.95
Período de Exposição $\gamma$	15	20
Período de Infectância $\alpha$	30	35
Período de Recuperação $\delta$	40	45

Tabela 4.1: Parâmetros utilizados nas simulações executadas.

A Tabela 4.2 apresenta as dimensões das matrizes utilizadas no sistema de simulação para aproximar as geometrias dos lotes da quadra 445, ilustrada nas Figuras 3.1 e 3.2 da seção 3. A quadra 445 da cidade de Cascavel/PR foi utilizada como estudo de caso ao problema, servindo como base à definição do ambiente computacional para execução das simulações. As dimensões apresentadas foram obtidas com auxílio da ferramenta SIMULA, utilizando-se os processos já apresentados e discutidos na seção 3.5.

	Lote	Quantidade de Linhas	Quantidade de Colunas
0	001	14	9
1	002-1	3	13
2	002-2	4	5
3	011	21	17
4	013	13	9
5	014	14	9
6	015	14	9
7	016	12	10
8	017	25	13
9	018	45	13
10	03B	19	34
11	03C	19	34
12	03E	10	11
13	03F	11	11
14	19A	45	13
15	19B-1	22	5
16	19B-2	9	21

Tabela 4.2: Dimensões das matrizes utilizadas para representação dos lotes.

Os tempos de execução e *speedups* obtidos para as implementações convencionais e *bitstring* são apresentados na Tabela 4.3. O *speedup*, no contexto deste trabalho, indica o aumento de desempenho entre a execução de simulações de forma sequencial, utilizando a CPU, e de forma paralela, utilizando a GPU. Especificamente, o *speedup* indica o fator de redução no tempo de execução, quando a execução de uma simulação é realizada em GPU.

Implementação	Tempo de Execução em CPU (s)	Tempo de Execução em GPU (s)	Speedup
Convencional	14401	1567	9.19
<i>Bitstring</i>	15392	1301	11.83

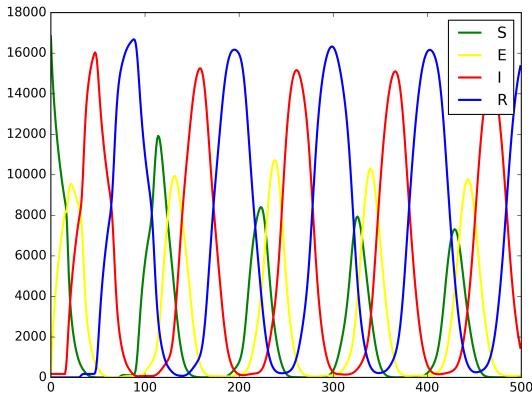
Tabela 4.3: Tempo de execução e *speedups* obtidos para as diferentes implementações realizadas.

Os resultados apresentados foram obtidos por meio da execução de simulações, utilizando a API CUDA à paralelização do sistema de simulação. Especificamente, para a execução de simulações na GPU utilizou-se uma placa gráfica Nvidia Tesla K20, que conta com 2496 *cuda cores* com frequência de 706 MHz, 5 GB de memória e desempenho de 1.17 Tflops e 3.52 Tflops para ponto flutuante de precisão dupla e simples, respectivamente [Tesla 2016]. Para a execução de simulações em CPU, utilizou-se uma máquina com processador Intel Xeon E5-2620, que conta com 6 *cores* e 12 *threads* com frequência de 2 GHz, dispondo de 128 GB de memória. O computador em que as simulações foram executadas possui o sistema operacional Ubuntu 16.04. Utilizou-se *scripts* escritos em *shell script* [Script 2016] para o controle da execução das simulações e obtenção dos tempos de execução. A máquina utilizada foi reiniciada antes do início da execução dos testes e somente os processos essenciais ao sistema operacional foram mantidos em execução.

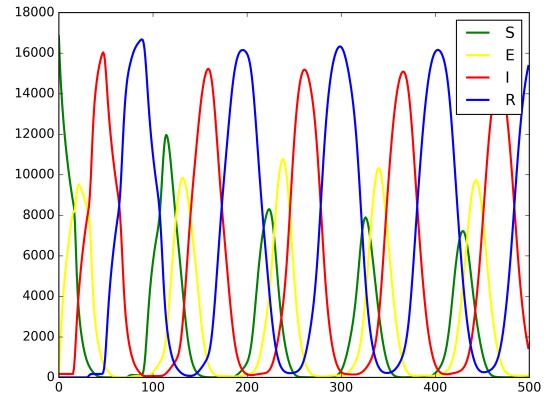
Pode-se observar que, para a implementação convencional, o tempo de processamento foi reduzido em aproximadamente 9 vezes. Já para a implementação em *bitstring*, o tempo de processamento foi reduzido em aproximadamente 12 vezes. É interessante notar que, os tempos de execução e *speedups* têm valores aproximados entre as implementações convencionais e *bitstring*. Portanto, pode-se concluir que, no caso da implementação realizada, não houve redução significativa no tempo de execução somente com a aplicação da técnica de *bitstring*.

A Figura 4.5 ilustra os resultados obtidos para a dinâmica epidemiológica utilizando os parâmetros apresentados na Tabela 4.1. Estes gráficos foram obtidos por meio de pós-processamento, externo ao *software SIMULA*, aplicado aos arquivos de saídas das simulações,

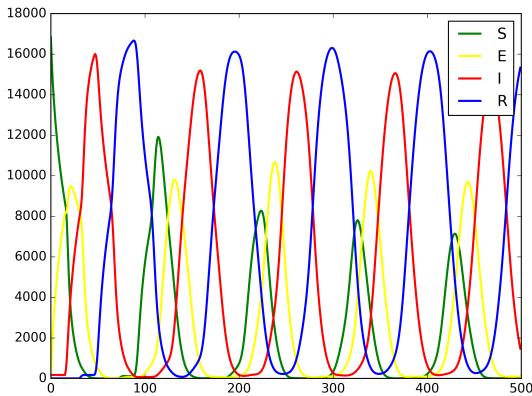
utilizando um *script* implementado na linguagem de programação Python [Python 2016]. Com estas saídas é possível observar o comportamento das populações de indivíduos suscetíveis, expostos, infectantes e recuperados durante o tempo de execução da simulação, representados pelas linhas nas cores verde, amarelo, vermelho e azul, respectivamente. Pode-se notar ainda que os resultados das diferentes implementações apresentadas nas figuras possuem dinâmicas assemelhadas entre si. Em geral, as implementações têm resultados qualitativamente assemelhados, apresentando algumas poucas diferenças quantitativas, discutidas em sequência.



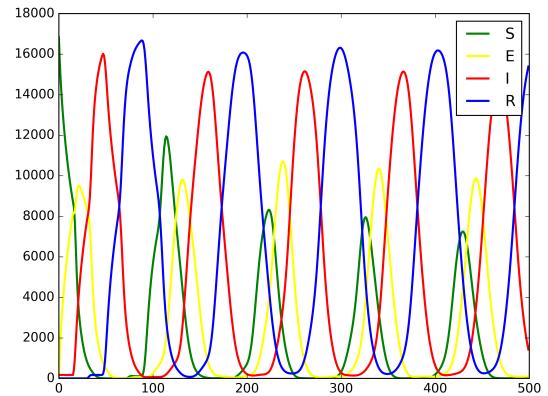
Implementação convencional em CPU



Implementação convencional em GPU



Implementação *bitstring* em CPU



Implementação *bitstring* em GPU

Figura 4.5: Gráficos das quantidades de indivíduos *versus* tempo de simulação

Inicialmente, as implementações CPU e GPU apresentavam resultados qualitativamente desontantes. Após a execução de testes verificou-se que esta diferença qualitativa estava relacionada aos mecanismos utilizados para a geração de números pseudo-aleatórios. Para as implemen-

tações em GPU, utiliza-se um gerador disponível na biblioteca *Curand* [Curand 2016], que gera números seguindo uma distribuição uniforme, enquanto que nas implementações em CPU eram utilizados geradores que não garantem tal comportamento, como a função *rand* da biblioteca *stdlib* [Rand 2016]. Para garantir a compatibilidade entre as duas implementações foram adotados nas implementações em CPU geradores de números pseudo-aleatórios que seguem uma distribuição uniforme, que estão disponíveis na biblioteca *random* da linguagem C++ [Random 2016].

Após a compatibilização das distribuições utilizadas para a geração de números pseudo-aleatórios foram confeccionados gráficos para visualizar os erros relativos entre as implementações realizadas. Os erros relativos entre duas implementações foram calculados para cada população, ciclo a ciclo, utilizando-se as expressões 4.1 e 4.2. Na expressão 4.1,  $CPU(i)$  e  $GPU(i)$  designam as soluções em CPU e GPU, no ciclo  $i$ , respectivamente. Na expressão 4.2,  $C(i)$  e  $B(i)$  designam as soluções convencionais e *bitstring*, no ciclo  $i$ , respectivamente.

$$E_R^{CPU, GPU} = \left| \frac{CPU(i) - GPU(i)}{CPU(i)} \right| \quad (4.1)$$

$$E_R^{C,B} = \left| \frac{C(i) - B(i)}{C(i)} \right| \quad (4.2)$$

A Figura 4.6 ilustra os erros relativos entre as implementações convencional e *bitstring* em CPU, a Figura 4.7 ilustra os erros relativos entre as implementações convencional e *bitstring* em GPU, a Figura 4.8 ilustra os erros relativos entre as implementações convencionais em CPU e GPU e a Figura 4.9 ilustra os erros relativos entre as implementações *bitstring* em CPU e GPU. As figuras que ilustram os erros relativos entre as implementações realizadas mostram que os resultados obtidos estão adequadamente próximos, independentemente da abordagem utilizada à representação dos indivíduos, convencional ou *bitstring*, e da execução em CPU ou GPU, apresentando erros menores do que 0.588%.

Pode-se observar ainda que existem diversos picos para as populações de indivíduos em todos os resultados. Eles acontecem pois as quantidades de indivíduos são muito próximas de zero, o que acarreta em um aumento de sensibilidade no cálculo do erro relativo entre as quantidades. É possível observar, por meio da análise da Figura 4.5 que, quando as quantidades

de uma determinada população se aproximam de zero em um ciclo de tempo, os erros relativos entre suas implementações aumentam aproximadamente neste mesmo ciclo.

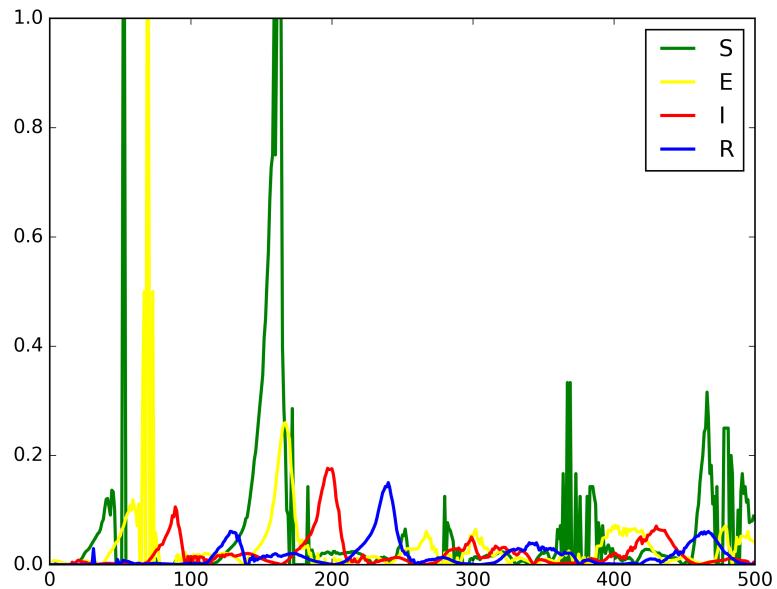


Figura 4.6: Gráficos dos erros relativos entre as implementações convencional e *bitstring* em CPU.

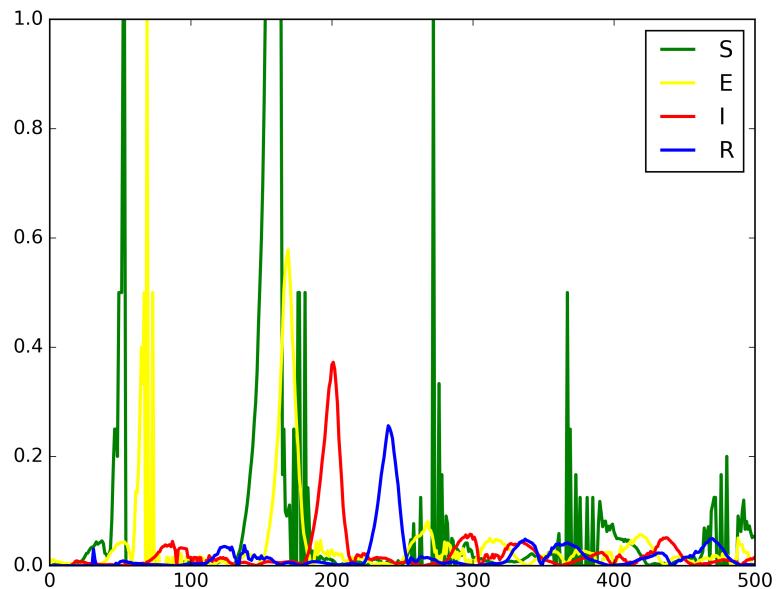


Figura 4.7: Gráficos dos erros relativos entre as implementações convencional e *bitstring* em GPU.

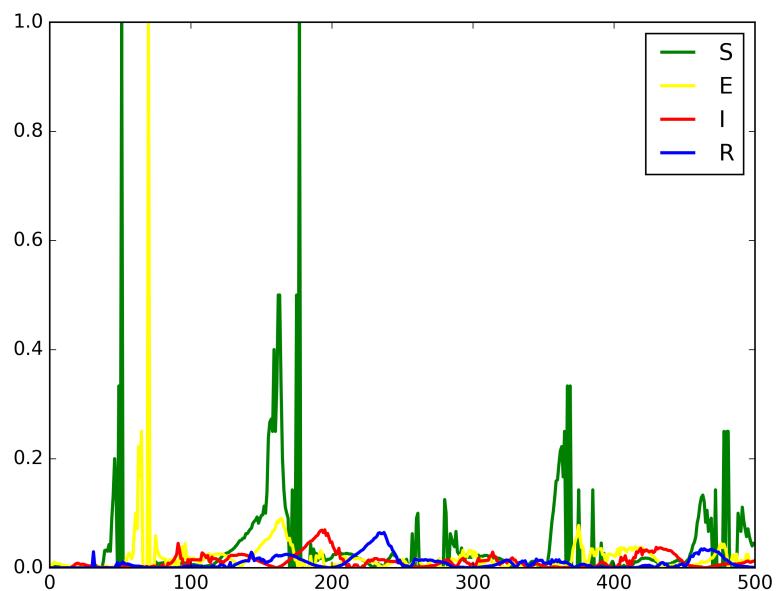


Figura 4.8: Gráficos dos erros relativos entre as implementações convencionais em CPU e GPU.

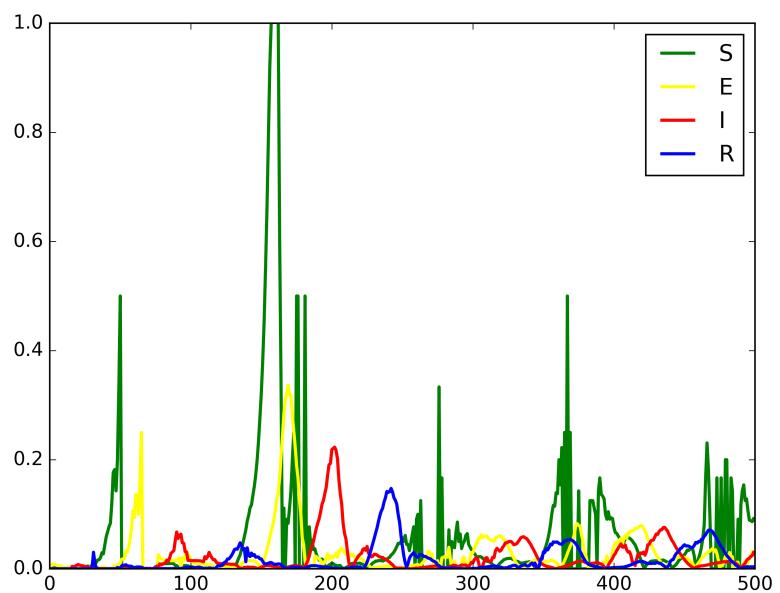


Figura 4.9: Gráficos dos erros relativos entre as implementações *bitstring* em CPU e GPU.

## 4.3 Dinâmicas Espaço-Temporais na Quadra 445.

As Figuras 4.10 e 4.11 mostram as saídas espaço-temporais obtidas por meio do *software* SIMULA para a quadra 445. Pode-se observar que as duas implementações, convencional e *bitstring* apresentam resultados assemelhados, não sendo idênticos devido a natureza estocástica dos processos de movimentação, contato e transição dos indivíduos. São apresentados, para cada implementação, o estado dos indivíduos na quadra, iniciando-se do ciclo 0, que corresponde à condição inicial, até o ciclo 100, em intervalos de 20 ciclos. Por meio das figuras é possível observar a evolução da dinâmica epidemiológica no domínio modelado.

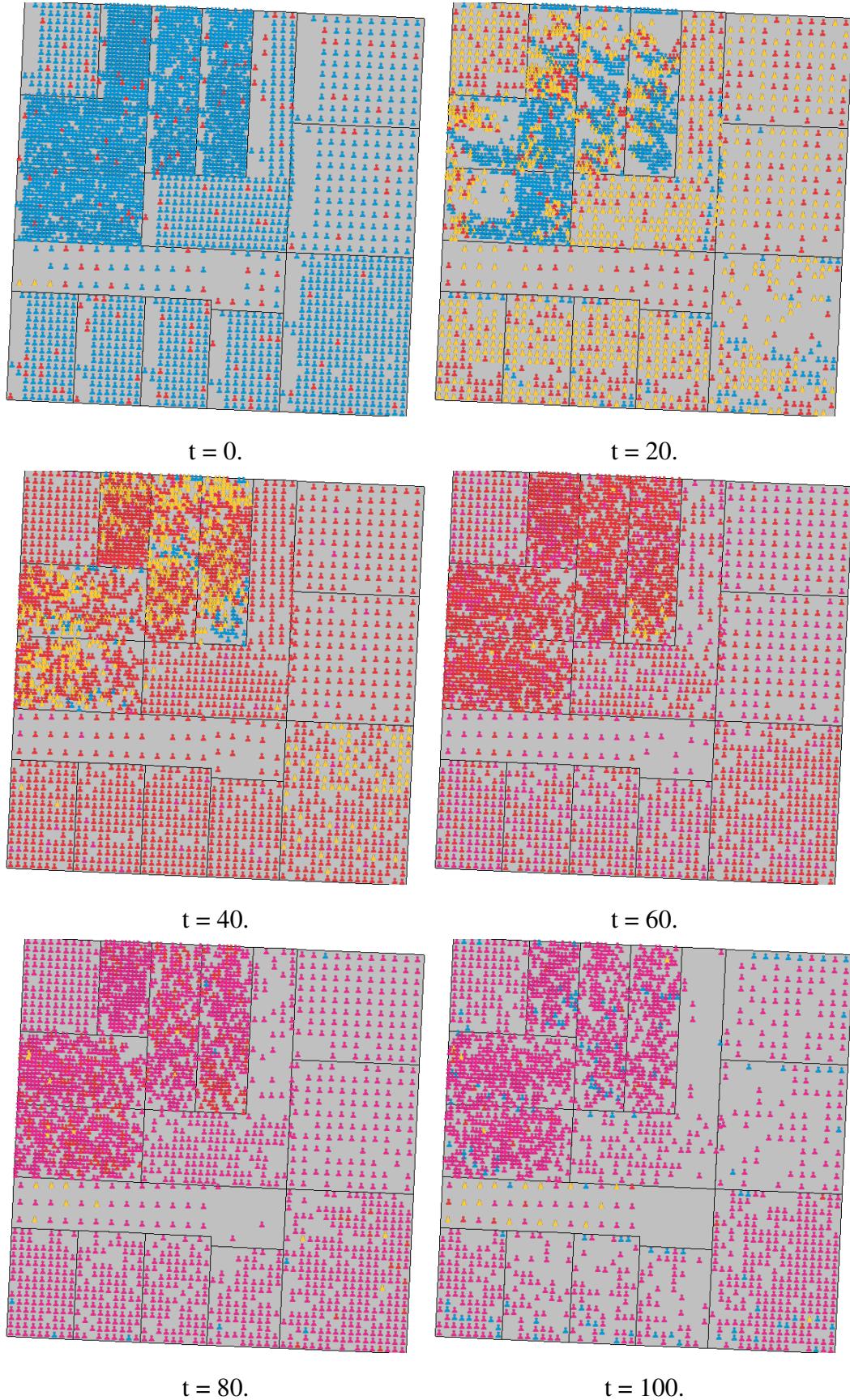


Figura 4.10: Dinâmica espaço-temporal para a implementação convencional.

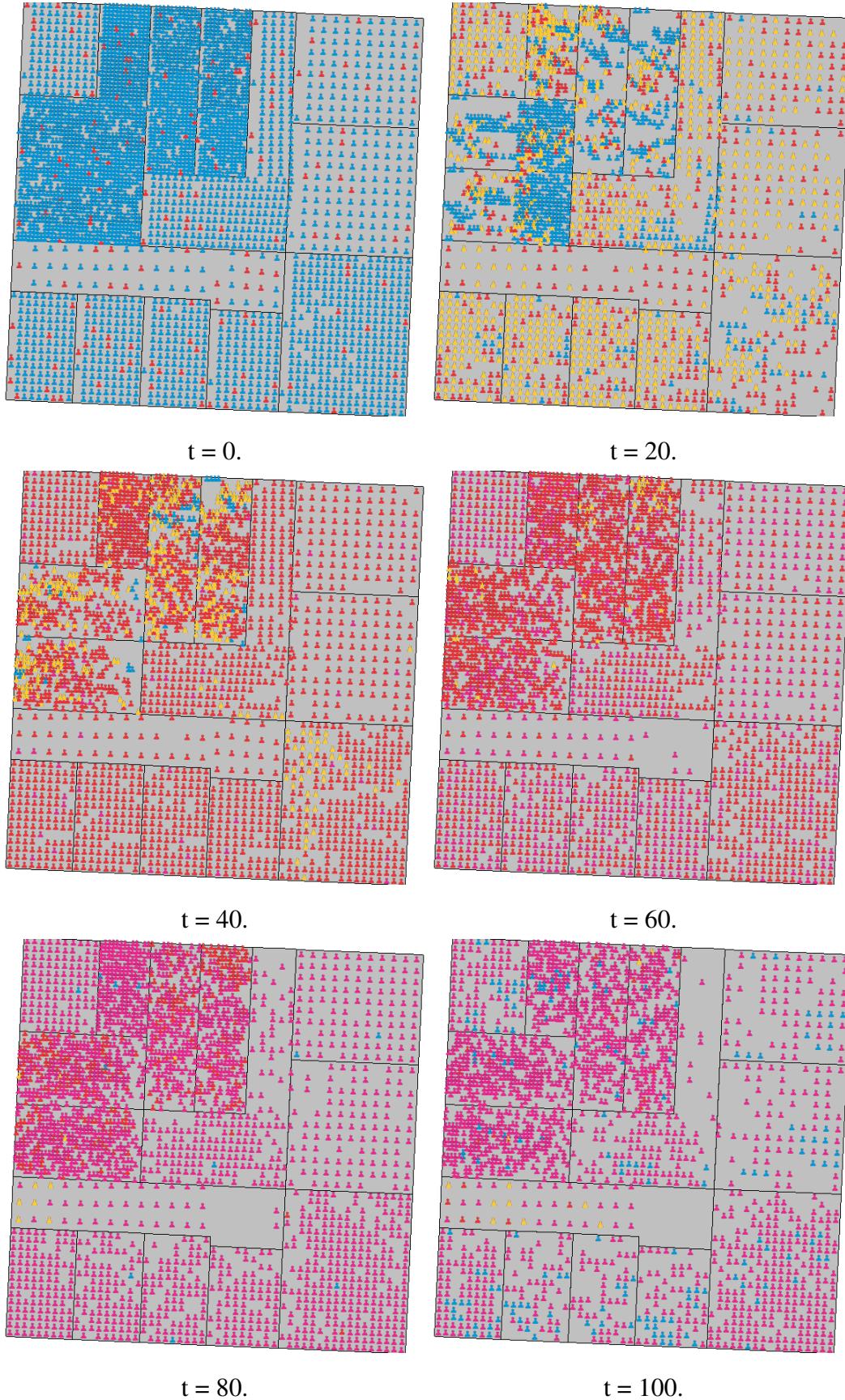


Figura 4.11: Dinâmica espaço-temporal para a implementação *bitstring*.

# Capítulo 5

## Conclusões

Neste trabalho foi apresentado um modelo computacional multiagente para simular a propagação de hipotéticas doenças baseadas em modelagem compartimental tipo SEIRS. Utilizou-se uma ferramenta computacional especificamente desenvolvida para a geração de um mapeamento georreferenciado da quadra 445 da cidade de Cascavel/PR, que serviu como ambiente para a execução de simulações. Foi proposto um modelo em *bitstring* para os agentes e operações relacionadas, facilitando o posterior emprego da biblioteca CUDA à paralelização do sistema de simulação. Por fim, apresentou-se os resultados obtidos com a execução de testes nas implementações realizadas, comparando-se os resultados obtidos entre as abordagens convencional e *bitstring*, e as execuções sequenciais e paralelas.

Conclui-se que o modelo multiagente implementado apresenta resultados interessantes, sobretudo sob o ponto de vista espaço-temporal, com dinâmica condizente com o esperado e mostrando a viabilidade da aplicação de modelos compartmentais à simulação de doenças hipotéticas sob áreas geográficas. O comportamento das populações de indivíduos suscetíveis, expostos, infectantes e recuperados, que podem ser observadas nos gráficos presentes na Figura 4.5, também foi de acordo com o esperado, conforme observado em trabalhos correlatos disponíveis na literatura.

O emprego da ferramenta SIMULA à geração do ambiente utilizado na simulação foi importante ao desenvolvimento do trabalho, sobretudo nas etapas de pré e pós-processamento, como na geração de arquivos de entrada para a simulação e visualização dos resultados, especialmente em saídas georreferenciadas. Notou-se que há uma grande dificuldade no tratamento e disponibilização de dados georreferenciados, especialmente por exigirem na maioria das vezes, de algoritmos e tecnologias específicas no seu armazenamento e processamento.

A aplicação da técnica de *bitstring* mostrou-se relevante, auxiliando na paralelização do sistema de simulação e diminuindo a quantidade de dados copiada entre CPU e GPU. Tal metodologia viabiliza ainda uma maneira concisa e uniforme de armazenamento e manipulação dos dados, sendo interessante sua aplicação no desenvolvimento de sistemas que modelem comportamentos individuais.

Por fim, a aplicação da biblioteca CUDA para a paralelização das implementações realizadas mostrou-se interessante por reduzir expressivamente o tempo de execução das simulações, o que é desejável na execução de simulações com maior porte, com maior quantidade de indivíduos, em ambientes maiores e com modelagens mais complexas às dinâmicas epidemiológicas, que são situações mais próximas da realidade que deseja-se simular. Destaca-se ainda que a implementação de sistemas em GPU, embora facilitada por bibliotecas específicas como o CUDA, ainda apresenta diversas dificuldades e relativa complexidade, sobretudo quanto à adequação dos dados para viabilizar seu processamento em paralelo e transferência para a GPU, que pode demandar um longo período de planejamento e desenvolvimento. Especificamente neste trabalho foi necessária a vetorização de todas as estruturas de dados para viabilizar sua cópia para a GPU. Este processo pode não ser trivial, especialmente se a implementação que deseja-se paralelizar em GPU fizer uso de algoritmos complexos e estruturas de dados dinâmicas, como aquelas disponíveis em linguagens de programação de alto nível, como a biblioteca *Standard Template Library*, STL, do C++ [STL 2016]. Adicionalmente foi necessária a adequação das rotinas paralelizadas, exigindo a eliminação de dependências que inviabilizam o processamento em paralelo, como as temporais, e organização da sequência de processamento dos dados de forma a viabilizar a computação por diferentes *threads*, garantindo a consistência dos dados.

Sugere-se como trabalhos futuros:

- Desenvolvimento de modelos mais sofisticados para propagação de doenças, que incluem o agente vetor, viabilizando modelar comportamentos individuais como a movimentação na busca por acasalamento e alimento. Pode-se ainda modelar a aplicação de controles químicos, biológicos e mecânicos sobre a população de vetores.
- Aplicação da metodologia *bitstring* sobre modelagens mais complexas, com agentes que apresentem mais atributos, que necessitem do uso de palavras computacionais com maior quantidade de *bits*.

- Definição de um modelo *bitstring in place*, que realize as operações de configuração de atributos utilizando uma quantidade mínima de variáveis auxiliares, reduzindo o consumo de memória do computador.
- Emprego e comparação de diferentes técnicas para paralelização do sistema de simulação, como por exemplo OpenMP, OpenMPI e outras APIs para programação em GPU.
- Estudo mais aprofundado sobre os parâmetros utilizados no modelo, sobretudo em suas influências sobre a dinâmica epidemiológica.
- Utilização de outros ambientes georreferenciados como base à simulação.
- Aprimoramento das estruturas de dados utilizadas, objetivando descartar a necessidade de vetorização. Pode-se estudar como realizar o uso ou mapeamento das estruturas de dados disponíveis na biblioteca STL do C++ com aquelas presentes na biblioteca *Thrust* do CUDA [Thrust 2016].

# Referências Bibliográficas

- [Aaby, Perumalla e Seal 2010] AABY, B. G.; PERUMALLA, K. S.; SEAL, S. K. Efficient simulation of agent-based models on multi-gpu and multicore clusters. Tennessee, EUA, 2010.
- [Alves e Gagliardi 2006] ALVES, D.; GAGLIARDI, H. F. *Técnicas de Modelagem de Processos Epidêmicos e Evolucionários*. 1. ed. São Paulo: Sociedade Brasileira de Matemática Aplicada e Computacional, 2006.
- [Bortoluzzi 2016] BORTOLUZZI, D. V. D. *Desenvolvimento de Ferramenta de Software para Manipulação, Disponibilização e Visualização de Dados Georreferenciados*. Dissertação (Monografia) — Universidade Estadual do Oeste do Paraná - UNIOESTE, Cascavel - PR, Fevereiro 2016.
- [Chwif e Medina 2014] CHWIF, L.; MEDINA, A. C. *Modelagem e Simulação de Eventos Discretos: Teoria e Aplicações*. [S.l.]: Elsevier Brasil, 2014.
- [Cook 2013] COOK, S. *Cuda Programming - A Developer's Guide to Parallel Computing with GPUs*. 1. ed. EUA: Elsevier, 2013.
- [Curand 2016] CURAND. Curand. 2016. Consultado na INTERNET: <http://docs.nvidia.com/cuda/curand/device-api-overview.html#axzz4O1MAZH5s>, 2016.
- [DAGHLIAN 2008] DAGHLIAN, J. *Lógica e Álgebra de Boole*. 4. ed. [S.l.]: Atlas, 2008.
- [Daley e Gani 1999] DALEY, D. J.; GANI, J. *Epidemic Modelling - An Introduction*. 1. ed. Reading: Cambridge University, 1999.
- [DirectX 2016] DIRECTX. DirectX. 2016. Consultado na INTERNET: <https://pt.wikipedia.org/wiki/DirectX>, 2016.

- [Fuks e Lawniczak 2001] FUKS, H.; LAWNICZAK, A. T. Individual-based lattice model for spatial spread of epidemics. *Discrete Dynamics in Nature and Society*, v. 6, p. 181–200, 2001.
- [Holvenstot 2014] HOLVENSTOT, P. *GPU-Accelerated Influenza Simulations for Operational Modeling*. Dissertação (Dissertação de Mestrado) — Western Michigan University, Michigan, EUA, 2014.
- [Holvenstot, Prieto e Doncker] HOLVENSTOT, P.; PRIETO, D.; DONCKER, E. de. Gpgpu parallelization of self-calibrating agent-based influenza outbreak simulation. Michigan, EUA.
- [Kaizer et al. 2016] KAIZER, W. L. et al. Solução paralela de um modelo baseado em indivíduos para simulação computacional da propagação de doenças. *Medianeira in Technology - MEDITEC*, Universidade Tecnológica Federal do Paraná - UTFPR, Medianeira, PR, 2016.
- [Luiz 2012] LUIZ, M. H. R. *Modelos Matemáticos em Epidemiologia*. Dissertação (Dissertação de Mestrado) — Universidade Estadual Paulista "Júlio de Mesquita Filho", Rio Claro - SP, 2012.
- [Lysenko, D'Souza e Rahmani 2007] LYSENKO, M.; D'SOUZA, R.; RAHMANI, K. A framework for megascale agent based model simulations on the gpu. *Parallel and Distributed Computing*, Michigan, EUA, 2007.
- [McNamara e Zanetti 1988] MCNAMARA, G. R.; ZANETTI, G. Use of the boltzmann equation to simulate lattice-gas automata. *Physical Review Letters*, EUA, v. 61, n. 20, p. 2332–2335, 1988.
- [Medronho et al. 2008] MEDRONHO, R. A. et al. *Epidemiologia*. 2. ed. Reading: Atheneu, 2008.
- [Oliveira 2001] OLIVEIRA, I. C. de. *Aplicação de Data Mining na Busca de um Modelo de Prevenção da Mortalidade Infantil*. Dissertação (Dissertação de Mestrado) — Universidade Federal de Santa Catarina, Florianópolis - SC, 2001.
- [OpenGL 2016] OPENGL. *OpenGL*. 2016. Consultado na INTERNET:  
<https://www.opengl.org/>, 2016.

[OpenMP 2016]OPENMP. *OpenMP*. 2016. Consultado na INTERNET: <http://openmp.org/wp/>, 2016.

[OpenMPI 2016]OPENMPI. *OpenMPI*. 2016. Consultado na INTERNET: <https://www.openmpi.org/>, 2016.

[Paiva e Nepomuceno 2013]PAIVA, B. de P. O.; NEPOMUCENO, E. G. Análise de estabilidade e controle de doenças infecciosas por meio de um modelo compartmental. *Sociedade Brasileira de Matemática Aplicada e Computacional*, v. 1, 2013.

[Paixão 2012]PAIXÃO, C. A. *Modelo de Bitstring para Estudo da Propagação da Dengue*. Tese (Tese de Doutorado) — Universidade Federal de Lavras, Lavras, MG, 2012.

[PostGIS 2016]POSTGIS. *PostGIS*. 2016. Consultado na INTERNET: <http://postgis.net/>, 2016.

[PostgreSQL 2016]POSTGRESQL. *PostgreSQL*. 2016. Consultado na INTERNET: <https://www.postgresql.org/>, 2016.

[Python 2016]PYTHON. *Python*. 2016. Consultado na INTERNET: <https://www.python.org/>, 2016.

[Rand 2016]RAND. *Rand*. 2016. Consultado na INTERNET: <http://www.cplusplus.com/reference/cstdlib/rand/>, 2016.

[Random 2016]RANDOM. *Random*. 2016. Consultado na INTERNET: <http://www.cplusplus.com/reference/random/?kw=random>, 2016.

[Russel e Norvig 2003]RUSSEL, S.; NORVIG, P. *Artificial Intelligence - A Modern Approach*. 2. ed. Reading: Prentice Hall, 2003.

[Sanders e Kandrot 2010]SANDERS, J.; KANDROT, E. *Cuda By Example - An Introduction to General-Purpose GPU Programming*. 1. ed. EUA: Addison-Wesley, 2010.

[Script 2016]SCRIPT, S. *Shell Script*. 2016. Consultado na INTERNET: [https://pt.wikipedia.org/wiki/Shell\\_script](https://pt.wikipedia.org/wiki/Shell_script), 2016.

[Shapefile 2016]SHAPEFILE. *Shapefile*. 2016. Consultado na INTERNET: <https://doc.arcgis.com/en/arcgis-online/reference/shapefiles.htm>, 2016.

- [Solutions 2016]SOLUTIONS, L. *Language Solutions*. 2016. Consultado na INTERNET:  
<https://developer.nvidia.com/language-solutions>, 2016.
- [SQL 2016]SQL. *SQL*. 2016. Consultado na INTERNET:  
<http://www.w3schools.com/sql/default.asp>, 2016.
- [STL 2016]STL. *STL*. 2016. Consultado na INTERNET:  
<http://www.cplusplus.com/reference/stl/?kw=stl>, 2016.
- [Tesla 2016]TESLA, N. *Nvidia Tesla*. 2016. Consultado na INTERNET:  
<http://www.nvidia.com.br/object/workstation-solutions-tesla-br.html>, 2016.
- [Thrust 2016]THRUST. *Thrust*. 2016. Consultado na INTERNET:  
<http://docs.nvidia.com/cuda/thrust/#axzz4NWvyVCua>, 2016.
- [White, Rey e Sanchez 2007]WHITE, S. H.; REY, A. M. del; SANCHEZ, G. R. Modeling epidemics using cellular automata. *Applied Mathematics and Computation*, v. 186, n. 1, p. 193–202, 2007.
- [Willem 2015]WILLEM, L. *Agent-Based Model For Infectious Disease Transmission*. Tese (Tese de Doutorado) — Antwerpen University, Antwerpen, Belgium, 2015.
- [Wooldridge 2016]WOOLDRIDGE, M. *Intelligent Agents*. 2016. Consultado na INTERNET:  
<http://www.cs.ox.ac.uk/people/michael.wooldridge/pubs/mas99.pdf>, 2016.
- [Wooldridge e Jennings 1995]WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents - theory and practice. *The Knowledge Engineering Review*, England, v. 10, n. 2, p. 115–151, 1995.
- [Yang 2001]YANG, H. M. *Epidemiologia Matemática - Estudo dos Efeitos da Vacinação em Doenças de Transmissão Direta*. 1. ed. Reading: Unicamp, 2001.