UNIVERSIDADE DE SÃO PAULO
FACULDADE DE FILOSOFIA, CIÊNCIAS E LETRAS DE RIBEIRÃO PRETO
PROGRAMA DE FÍSICA APLICADA À MEDICINA E BIOLOGIA

Novos Resultados nas Caminhadas Deterministas Parcialmente Autorepulsivas em Meios Aleatórios Obtidos com o Gerenciamento Numérico da Memória dos Caminhantes

Wilnice Tavares Reis Oliveira

Ribeirão Preto
2010

UNIVERSIDADE DE SÃO PAULO
FACULDADE DE FILOSOFIA, CIÊNCIAS E LETRAS DE RIBEIRÃO PRETO
PROGRAMA DE FÍSICA APLICADA À MEDICINA E BIOLOGIA

Novos Resultados nas Caminhadas Deterministas Parcialmente Autorepulsivas em Meios Aleatórios Obtidos com o Gerenciamento Numérico da Memória dos Caminhantes

Wilnice Tavares Reis Oliveira

Tese submetida ao Programa de Pós-Graduação em Física Aplicada à Medicina e Biologia da Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto da Universidade de São Paulo como parte dos requisitos para a obtenção do título de Doutor em Física Aplicada à Medicina e Biologia.

Orientador: Prof. Dr. Alexandre Souto Martinez

Ribeirão Preto, janeiro de 2010.

Oliveira, W T R

Novos Resultados nas Caminhadas Deterministas Parcialmente Autorepulsivas em Meios Aleatórios Obtidos com o Gerenciamento Numérico da Memória dos Caminhantes / Wilnice Tavares Reis Oliveira; orientador Prof. Dr. Alexandre Souto Martinez.

– Riberão Preto/SP, 2010.

109 f.

Tese (Doutorado – Programa de Pós-Graduação em Física Aplicada à Medicina e Biologia) – Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto da Universidade de São Paulo.

Sistemas dinâmicos, caminhada do turista, caminhada com memória, meios desordenados, lista hierárquica, distribuição conjunta, distribuição de tempos de transiente, distribuição de período de atratores, memória crítica, atratores proibidos.

Banca Examinadora:

_____

_____

_____

_____

_____

Dedicatória

Ao meu marido Luís Cézar, pelo seu estímulo, paciência
e indefinível carinho, sempre a me apoiar.
Aos meus pais, irmãos e cunhados, pela paciência despendida,
e pelo incentivo nas horas difíceis.
Aos meus queridos sobrinhos, pelo indubitável carinho
dedico o presente trabalho.

Agradecimentos

Agradeço à Fundação de Amparo à Pesquisa do Estado de Mato Grosso que proveu os recursos que possibilitaram o trabalho, realizado na Universidade de São Paulo (USP) em Ribeirão Preto, ao Instituto Federal de Educação, Ciência e Tecnologia de Mato Grosso (IFMT) que apoiou a concretização da pesquisa, aos meus colegas de trabalho do Departamento de Informática (DAI), onde venho ensinando e onde desenvolvemos um esforço conjunto para prover o conhecimento.

Agradecimento especial ao meu orientador, Alexandre Souto Martinez pelo acolhimento e paciência, pelos diversos conselhos valiosos e pelo interesse e amizade com que sempre se dispôs a me orientar.

Agradeço também à Antonio Carlos Roque da Silva Filho, Antonio Caliri, Carlos Alberto Pelá e Marco Antônio de Almeida, meus professores no programa de pós-graduação, que me apresentaram um universo de informações e experiências científicas.

Agradeço aos professores Renato Tinós, Cléver Ricardo Guareis de Farias e Luiz Otávio Murta Junior, pela participação na banca de defesa da minha qualificação e pelas críticas sinceras.

Destaco também um agradecimento aos professores Osame Kinouchi Filho, Nelson Augusto Alves, Marcelo Mulato, Evandro Eduardo Seron Ruiz, Adilton Oliveira Carneiro, Adelaide de Almeida, Rafael Rosales Mitrowsky, Fernando Pigeard de Almeida Prado, pela iluminação em alguns momentos onde apresentei minhas suposições sobre a pesquisa, pelas aulas que me permitiram estar presente, como ouvinte apenas, e pela atenção que despendida em qualquer momento.

Agradeço aos meus colegas do curso, pelos comentários enriquecedores e pela grandiosa amizade conquistada.

Agradeço a todos os meus colegas do laboratório, também pela amizade adquirida e pelas boas horas de descontração compartilhadas.

Agradeço aos funcionários do departamento, pelo apoio em todos os sentidos.

“Tenha em mente que tudo que você aprende na escola é trabalho de muitas gerações. Receba essa herança, honre-a, acrescente a ela e, um dia, fielmente deposite-a nas mãos dos seus filhos.”

Albert Einstein.

Resumo

Podemos considerar a caminhada determinista do turista, como um processo do tipo dinâmico, que influi sobre uma rede composta de N pontos. Os pontos são gerados de maneira aleatória, no espaço euclidiano d dimensional. Um caminhante partindo de um ponto qualquer, no meio desordenado, se movimenta seguindo uma regra determinista de ir para o ponto mais próximo que não tenha sido visitado nos últimos μ passos anteriores. Cada uma das trajetórias, geradas através dessa dinâmica, possui uma parte inicial não periódica de t passos, denominada de transiente, e uma parte final, periódica de p passos, denominada de atrator. Neste estudo obtemos resultados numéricos interessantes, que serviram para explicar a formação de determinados atratores periódicos, na caminhada determinista do turista unidimensional, bem como, a não formação de atratores cujos períodos $2\mu+1$, $2\mu+2$ e $2\mu+3$, não são constituídos. Também neste trabalho, uma nova implementação na estrutura de armazenamento de dados, no modelo numérico do turista, nos permitiu obter algumas distribuições estatísticas para a caminhada, com valores de memória μ muito acima do que se tinha alcançado anteriormente. Com estes resultados verificamos a eficiência da estrutura proposta, e avançamos o conhecimento a cerca do comportamento do turista em sistema da ordem de N.

Palavras-chave: Sistemas dinâmicos, caminhada do turista, caminhada com memória, meios desordenados, lista hierárquica, distribuição conjunta, distribuição de tempos de transiente, distribuição de período de atratores, memória crítica, atratores proibidos.

Abstract

We consider the deterministic tourist walk, as a dynamic process, which influences a network composed of N points. The points are randomly generated, in a euclidean space d dimensional. A walker starting to walk from any point of that medium, moves following the deterministic rule of always going to the nearest point that has not been visited in the last μ steps. Each trajectories generated by dynamic, has an initial non-periodic part of t steps, called transient and a final periodic part of p steps, called attractor. In this study we obtain interesting numerical results, which served to explain the formation of the periodic attractors specific in deterministic walk in one-dimensional space and, no formation of the attractors whose periods $2\mu+1$, $2\mu+2$ and $2\mu+3$ are not made. In this work, we propose a new implementation of the structure for storing data, the numerical model of the tourist, has allowed us to obtain some statistical distributions for the walk with a memory value μ over and above what had been achieved previously. With these results, we verifed the efficiency of the HL structure proposed, and advance knowledge about the behavior of the tourist walk in the order of N.

Keywords: Dynamical systems, tourist walk, walk with memory, disordered media, hierarchical list, joint distribution, transient time distribution, attractor period distribution, critical memory, prohibited attractors.

Sumário

Lista de Símbolos

N : Número de pontos do meio;

d : Dimensionalidade do meio;

μ : Memória do turista;

t : Tempo de transiente;

p : Período do atrator;

$P^N_{\mu,d}(t)$: Distribuição de probabilidade de tempos de transiente em sistemas com N pontos e turista com memória μ em d dimensões;

$P^N_{\mu,d}(p)$: Distribuição de probabilidade de atratores de período p em sistemas com N pontos e turista com memória μ em d dimensões;

d_{ij} : Distância entre os pontos i e j;

$D_{N\times N}$: Matriz quadrada da distância de dimensão $N\times N$;

$V_{N\times\mu+1}$: Matriz de vizinhança de dimensão $N\times(\mu+1)$;

$P(k_{out})$: Distribuição de probabilidade de arcos que saem de um vértice;

$\delta_{i,j}$: Delta de Kronecher;

$E(k_{in})$: Média de arcos que chegam em um vértice;

$S^N_{\mu,d}(t,p)$: Distribuição conjunta de tempos de transiente e períodos de um atrator, em sistemas com N pontos e turista com memória μ em d dimensões;

μ_c : Memória crítica responsável pelo ponto de transição no modelo do turista com memória para sistemas unidimensionais;

cac : Condições abertas de contorno;

cpc : Condições periódicas de contorno;

HL : Lista Hierárquica.

Lista de Figuras

Lista de Tabelas

1 Introdução

É bastante plausível e interessante iniciarmos essa discussão com um breve histórico a respeito do advento determinista no meio científico, que teve inúmeros seguidores, entre os quais destacamos o Marquês de Laplace. No seu trabalho sobre a Mecânica Celeste, Pierre Laplace supunha que a natureza era regida por leis inteiramente deterministas e assim deixou-se incentivar pelos efeitos produzidos pelas pertubações dos corpos celestes, no sistema solar. Diante desse paradigma, Laplace investigando o comportamento dos corpos, propôs métodos para se calcular o caminho percorrido pelos planetas em volta do sol, descrevendo assim, a evolução do sistema [1].

Em uma de suas percepções sobre os fenômenos naturais, Laplace defendeu a seguinte proposição: "Devemos portanto considerar, o estado presente do universo como o efeito de seu estado anterior, e como a causa daquele que seguirá."[2], ou seja, dada as condições iniciais de um sistema, a evolução deste poderia ser calculada, baseando-se no estado de tempo anterior. Dessa forma, nada seria incerto, e no universo não haveria lugar para o imprevisível. Apesar de todo amparado intelectual, a idéia de um universo determinista, não agradou muitos estudiosos. Tal questionamento, deu origem a outras ciências, mais especificamente, a ciência do caos. Importante lembrar que o caos pode ocorrer também em sistemas regidos por leis simples, ou seja, muitos fenômenos na natureza, cuja evolução se dá através de uma dinâmica determinista, não necessariamente precisa ser previsto.

Dando sequência as descobertas de Laplace, o matemático Henri Poincaré mostrou que o problema gravitacional dos três corpos, apresentava um estado estacionário não-periódico, o que caracterizaria um sistema caótico. Outro trabalho, também muito relevante, o qual popularizou-se de Efeito borboleta, fora apresentado por Edward Lorenz, no seu interesse em decifrar a metereologia. Usando um modelo numérico de previsão do tempo, Lorenz descobriu um atrator estranho, após uma mudança, pouco notável, nas condições iniciais do seu sistema, e aos poucos a ordem que havia na dinâmica, foi dando lugar a formação de ciclos identificáveis, que se repetiam sem nunca ser precisamente igual [3]. Assim como o tempo, na natureza existe inúmeros sistemas que chamam a atenção pela sua aperiódicidade, e o fundamento determinita,

não ficou restrito apenas ao conjunto das forças que dirigem o universo, mas também, às formas de sociedade humana também são influenciadas por essa conjectura, e dessa maneira são estudadas as epidemias que vão e vem, entre os povos, o fluxo, que sobe e desce, do mercado financeiro, etc.

Caberia ainda aqui, uma vasta discussão sobre a teoria determinista, uma vez que o *determinismo laplaciano*, como ficou conhecida, promove grande contestação intelectual a respeito do que pode ser previsível ou não. Porém, queremos deixar claro que o objetivo deste contexto, é apenas o de aduzir a origem do determinismo (no sentido puro e simples) e do célebre precursor dessa teoria.

Depois de expormos uma idéia do determinismo com os sistemas que evoluem no tempo, faremos uma introdução a dois tipos de modelos capazes de descrever o movimento de uma partícula em um fluído qualquer. Esses modelos classificados quanto a dinâmica imposta ao sistema, podem ser denominados de sistemas de caminhadas aleatórias ou sistemas de caminhadas deterministas. Modelar o comportamento de fenômenos naturais a partir de caminhadas aleatórias é uma prática já bastante comum cientificamente, ela pode envolver, desde sistmas que estudam o movimento de uma particula até sistemas macro, como os movimentos financeiros [4, 5, 6, 7], ao contrário das caminhadas deterministas, que devido a pouca exploração, ainda é um discusso bastante atual [8, 9, 10].

No estudo de caminhadas deterministas, estamos interessados em um modelo original, de caminhadas deterministas em meios aleatórios, notoriamente conhecido como a caminhada do turista. Neste modelo pretendemos conhecer não somente o comportamento do sistema na evolução do tempo, mas principalmente conhecer a influência das grandezas de controle, sobre o sistema e como elas se relacionam. No geral, varia-se todas as grandezas de controle, porém especificamente neste caso, trabalharemos com um aumento da ordem de N, na memória μ do caminhante, no modelo com dimensões $d > 1$.

A caminhada do turista é caracterizada por N pontos, distribuídos de forma aleatória e uniforme em um espaço d dimensional. A cada passo de tempo um caminhante situado em um dado ponto do meio, segue a seguinte regra deterministica: 'caminhar até o ponto vizinho mais próximo que não tenha sido percorrido nos últimos μ passos anteriores'. Embora simples de definir, este modelo determinista não é usualmente fácil de analisar, especialmente porque a medida que se aumenta os valores das grandezas de controle, as trajetórias se tornam complexas, e exponencialmente crescem a quantidade de informações que precisam ser armazenadas e analisadas.

Essas trajetórias são percorridas de acordo com a regra determinista, e de forma diferente

para cada um dos pontos de partida do caminhante. Porém, todas elas são formadas por uma parte de comprimento, denominada de transiente t, e outra parte de atratora, de tamanho p, onde o sistema estaciona. Apesar de não ser o nosso foco tratar uma outra variante do modelo, é interessante resaltar aqui, que uma vez modificada a regra do modelo, de modo que o caminhante se desloque para qualquer ponto do meio, com uma probabilidade proporcional à distância que o separa do ponto de destino, a caminhada então deixa de ser determinista e passa a ser considerada de caminhada estocástica do turista.

Hoje as técnicas de simulação computacional são ferramentas aliadas na resolução de problemas de cunnho científico. Numa busca para melhorar o armazenamento dos resultados numéricos, que extrapolavam a memória RAM (Random Access Memory) do computador, em tempo de execução, modificou-se a maneira de alocar esses dados, deixando a estrutura mais eficiente, embora não houvesse tido ganhos no esforço computacional, o que não nos causou nenhuma alteração.

O objetivo deste trabalho foi contemplar o comportamento do sistema determinista da caminhada do turista, considerando a janela de memória, do sistema, da ordem de N. Anteriormente, os dados de tempo de transiente e de peíodo de atratores, representados em uma matriz bidimensional ($t \times p$), crescia proporcionalmente, a medida que se ampliava o valor de N e de μ, devido ao armazenamento do maior transiente e do maior período formado em uma dada trajetória, isso proporcionava um consumo, realmente, muito elevado de memória RAM. Para isso, o emprego da nova estrutura de armazenamento de dados, foi indispensável, uma vez, que só a partir dessa mudança, foi possível analisar a dinâmica da caminhada, com valores muito acima do que já tínhamos considerado, nos revelando assim, particularidades do sistema, ainda não conhecidas.

Este trabalho de pesquisa está estruturado da seguinte forma: No Capítulo 2, apresentamos o modelo de caminhada determinista do turista, detalhando os aspéctos mais relevantes do modelo determinista. Abordamos algumas características que julgamos importante, como a determinação do meio aleatório, onde as caminhadas ocorrem, também como se dá a dinâmica do sistema, conhecendo os cálculos das distâncias e a formação da tabela de vizinhança, a propriedade de condição de contorno, que minimiza o efeito de borda em uma malha, e por último, um modelo de grafo orientado e ponderado do turista.

No Capítulo 3, antes de tudo, tratamos dos fundamentos básicos, necessários para a compreensão da estruturação do armazenamento dos dados no algoritmo do modelo, fazendo uma abordagem do problema enfretado e os tipos de recursos disponibilizados para a realização do processo de simulação numérico. Também discutimos as técnicas para armazenar informações binárias, destacando suas características de construção, bem como, os aspectos das duas for-

mas de alocação discutida no modelo do turista. Além disso, explanamos a forma como cada tipo de variável do sistema pode se representada em cada um dos tipos de registro de dados, ajustando-os tanto na estrutura de uma matriz de dados como na estrutura de uma lista de dados. Apresentamos também neste capítulo, todas as prerrogativas do uso de uma ou de outra estrutura de dados no modelo da caminhada do turista, de forma que elas possam ser comparadas. Destacamos também a grande vantagem da nova estrutura de dados implementada, no modelo, com o tipo lista, expondo os resultados numéricos, com o sistema variando nas dimensões $d = 1, 2$ e 3, onde foram avaliados o tempo de processamento computacional gasto e o tamanho da memória RAM consumida, quando a o sistema utiliza uma memória $\mu \gg 1$.

Iniciamos o Capítlo 4, nos referindo à discussões de trabalhos já publicados, no espaço unidimensional, bem como, expandimos estes estudos no modelo determinista, e verificamos que o comportamento de percolação no sistema, a partir de um valor crítico de memória, permaneceu na caminhada, realizada no meio com condições abertas de contorno. Observou-se que a partir de um dado valor crítico, todos as trajetórias se estabilizaram em ciclos de atratores de período $p \geq N$. Algo tão inusitado no comportamento das caminhadas foi verificado no modelo unidimensional, que nos motivou a buscar por um regime de transição de fase, na dinâmica do turista, também em outras dimensões, como investigamos em capítulos posteriores. Quando o valor da memória $\mu \approx N$, os pontos na paisagem se tornam todos igualmente atrativos, levando o caminhante a uma exploração global do meio, sugerindo dessa forma, a existência de um valor de memória crítica capaz de provocar notável mudança no sistema. Também no modelo unidimensional, detectamoos que a frequência de atratores $p_{min} = \mu + 1$ diminui a medida que a dinâmica evolui, e se aproxima do ponto crítico, algo que não ocorre em outras dimensões, devido à ausência da troca de regimes, de localizado para globalizado. E por último, verificamos que a unidimensionalidade, juntamente com o determinismo imposto no modelo, faz com que algumas combinações de pontos nunca ocorram, ocasionando a não existência de alguns períodos de atratores, o que nos faz imaginar que determinados passos, dados pelo caminhante, são proibidas.

De forma similar, no Capítulo 5, faremos algumas demonstrações dos resultados obtidos na caminhada determinista do turista no hiperespaço ($d > 1$). Iniciamos tratando do modelo no espaço bidimensional, enfatizando alguns resultados estatísticos interessantes para entendimento do estudo das trajetórias, como é o caso das distribuições de probabilidade de transiente t, das distribuições de probabilidade de atratores de período p e mais a probabilidade conjunta das duas grandezas t e p, que nos permite conhecer a caminhada como um todo, pelo fato de mostrar como tais grandezas se relacionam. Podemos afirmar que esse trabalho de pesquisa, teve uma tendência a se convergir, sobretudo para a investigação da formação dos períodos de

atratores. A alteração na estrutra de armazenamento de dados do modelo, no algoritmo da caminhada do turista (Capítulo 3), proporcionou a inserção de mais uma variável ou grandeza de estudo, com a qual pode-se investigar que pontos descrevem um determinado ciclo periódico.

Finalmente, no Capítulo 6 faremos as conclusões finais e indicações para os trabalhos futuros.

2 *Caminhadas Deterministas*

De acordo com a referência [3], sistemas dimâmicos são sistemas fora do equilíbrio, que evoluem e alteram o seu estado no domínio do tempo. O tempo pode ser classificado como uma variável contínua ou discreta. O estado do sistema pode ser compreendido como sendo o conjunto de valores das grandezas físicas, capaz de armazenar toda a informação histórica do mesmo em um determinado instante de tempo.

As pesquisas na área das caminhadas aleatórias já é bastante difundida entre os estudiosos, e alguns dos temas, no universo de trabalhos de seus pesquisadores, encontram-se bem representados segundo as referências [11, 12, 13, 14, 15, 16] que certamente não é uma lista exaustiva. Já no âmbito das caminhadas deterministas, fundamentadas na teoria dos sitemas dinâmicos, o seus efeitos ainda são pouco conhecidos, por ser esta uma área ainda pouco explorada, porém, vem depertando enorme interesse nos últimos tempos. Entre alguns trabalhos, do qual estamos cientes de que esse levantamento também não é exaustivo, se encontram [17, 18, 19, 20, 21]. Estes sistemas apresentam frequentemente um comportamento dinâmico muito complexo.

Estamos interessados no estudo de um modelo de caminhadas deterministas, originalmente conhecido como Caminhada do Turista[1]. Neste modelo, consideramos o tempo como uma variável discreta, enquanto a dinâmica determinista se desenvolve em um meio aleatório, uniformemente distribuído. Para o problema, já foram definidas medidas de probabilidade, descrevendo o comportamento estatístico em determinadas trajetórias [17]. Em vista disso, e com tais proposições, a idéia inicial é verificar outros comportamentos ainda não explorados na caminhada do turista, em dimensão maior que um.

[1] O nome Caminhada do Turista, foi proposto no ano de 2000, por Peter Stadler, em uma visita ao departamento de física e matemática (DFM) da Faculdade de Filosofia, Ciências e Letras da Universidade de São Paulo, pela analogia entre uma situação real de um turista explorar um determinado número de cidades, mutuamente próximas.

2.1 Caminhada do Turista

A caminhada determinista do turista é um modelo original de caminhada, cujo estado dinâmico evolui a intervalos discretos de tempo, e sua geometria pode ser representada por N pontos distribuídos aleatória e uniformemente, em um mapa d dimensional. De cada ponto[2] desse mapa, um caminhante se desloca, seguindo uma regra determinista. Esta regra estabelece que uma caminhada, iniciada de um ponto qualquer, se dê em direção ao ponto mais próximo que ainda não tenha sido visitado nos últimos μ intervalos de tempo. A memória μ mantén-se dentro do domínio $\mu \in [0, N-1]$.

De cada ponto, trajetórias distintas emergem no espaço d dimensional. Essas trajetórias são compostas por dois regimes bem definidos. Um transiente, ou seja, um regime aperiódico[3], que será representado por t, e outro regime periódico de período p, constituindo um conjunto de pontos regularmente visitados. Sabendo-se do caráter probabilístico da caminhada do turista, tornou-se importante o conhecimento de suas qualidades estatísticas. Dessa forma, em [22], foram definidas as medidas de distribuição conjunta $S_{\mu,d}(t,p)$, salientando as possívies condições iniciais. O cálculo da distribuição conjunta de transientes e de períodos (Eq.2.1), é obtida pelo produto da distribuição marginal de transiente $S_{\mu,d}(t)$ e a distribuição marginal de período $S_{\mu,d}(P)$.

$$S_{\mu,d}(t,p) = S_{\mu,d}(t).S_{\mu,d}(p) \tag{2.1}$$

A Fig. 2.1(a), apresenta uma caminhada do turista em um espaço bidimensional com $\mu = 2$, cuja trajetória exibe uma parte transiente com $t = 3$ e outra periódica com $p = 4$. Na Fig. 2.1(b), no mesmo espaço de configurações, outra caminhada definiu uma trajetória com transiente nulo e período $p = 3$. Enquanto que na Fig. 2.1(c), com $\mu = 3$, a trajetória gerada se assemelha a um polígono complexo com $p = 4$, cujo transiente também é nulo.

Outra característica da caminhada determinista do turista é a auto-repulsão, que pode se apresentar de maneira parcial ou total, dependendo do tamanho da memória do caminhante. Nesse sentido, a memória representa o tempo (em número de passos) necessário para que o ponto se torne propenso a visitação mais de uma vez (tempo refratário) [17].

No modelo da caminhada do turista, quando os valores de memória são fixados em N e $N-1$, por exemplo, no primeiro momento, teremos uma caminhada totalmente auto-repulsiva, onde a trajetória se conclui apenas com um estado transiente, sendo $t = N$ (Fig. 2.2), enquanto

[2] Um ponto também pode denotar cidade em um mapa ou sítio.

[3] Aperiódico indica que não há pontos em comum em um conjunto de pontos.

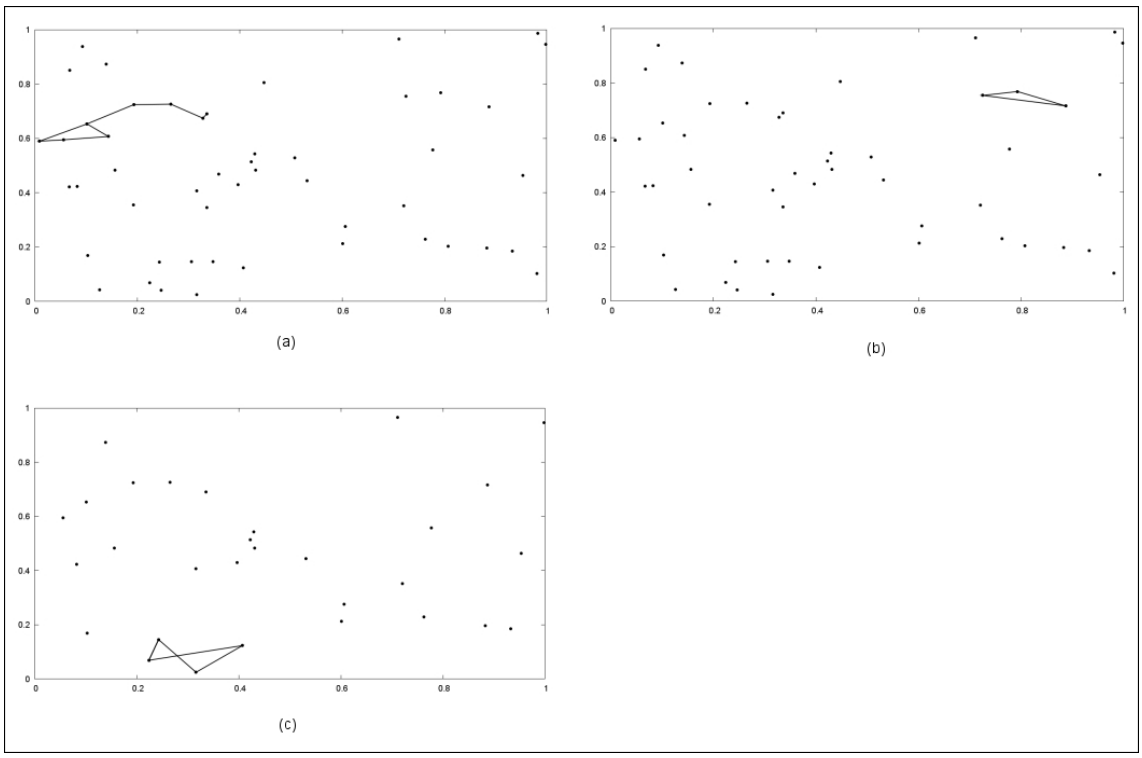


Figura 2.1: Exemplo de um meio desordenado, onde são apresentadas as caminhadas deterministas. (a): trajetória gerada no espaço bidimensional para $N = 50$ e $\mu = 2$, a qual apresenta uma parte transiente e uma periódica, (b): trajetória gerada no espaço bidimensional para $N = 50$ e $\mu = 2$, com transiente nulo, e (c): trajetória gerada no espaço bidimensional para $N = 30$ e $\mu = 3$, onde esta se assemelha a um polígono complexo, na qual o transiente também é nulo.

que, no segundo, a caminhada produzirá um atrator (ou ciclo) de período N, com transiente nulo (Fig. 2.3).

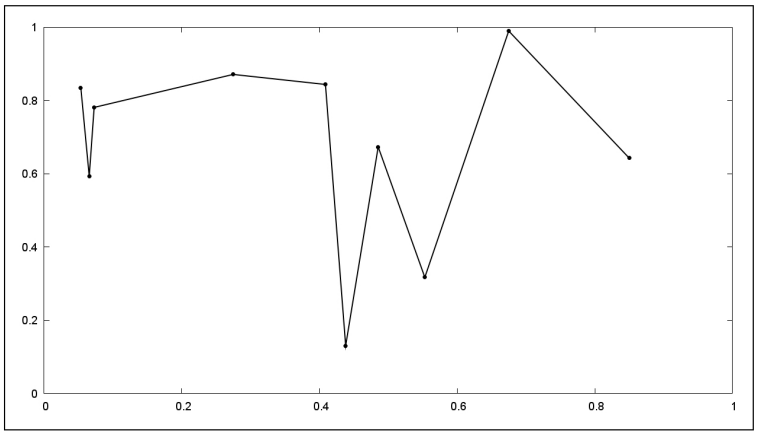


Figura 2.2: Caminhada do turista em um meio desordenado com $\mu = N = 10$.

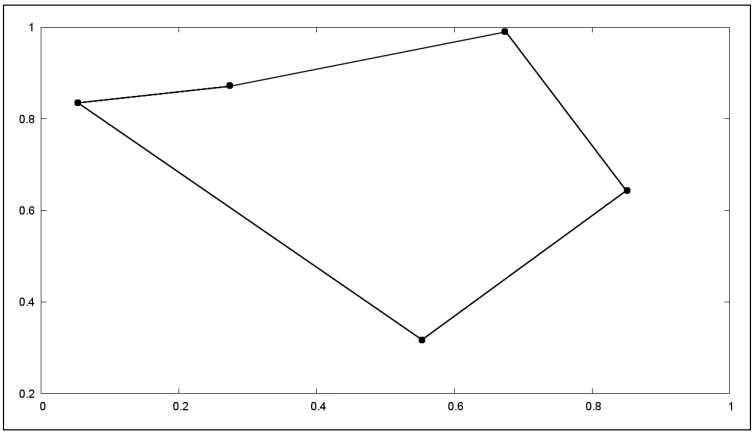


Figura 2.3: Caminhada do turista em um meio desordenado com $\mu = N - 1 = 4$.

2.1.1 Meio Desordenado

A modelagem numérica da caminhada do turista se divide em duas partes, uma estática que representa o meio desordenado e outra dinâmica que determina a própria caminhada. A primeira etapa no algoritmo é mensurar as distâncias entre os pontos, através da métrica euclidiana. Esses pontos são gerados aleatoriamente e representam o meio desordenado. Dessa forma, a distância D_{ij} entre quaisquer pares de pontos i e j é definida através da métrica:

$$D_{ij} = \sqrt{\sum_{k=1}^{d} \left[x_i^{(k)} - x_j^{(k)} \right]^2}, \tag{2.2}$$

onde $x_i^{(k)} \; \forall \; i \; e \; k \in \mathbb{N} / 1 \leq i \leq N \; e \; 1 \leq k \leq d$, em um espaço de d dimensões.

As coordenadas são geradas independentemente por uma função computacional pseudo-aleatória (a sequência numérica pode ser prevista, uma vez que se conhece a semente inicial) [23], em seguida, é criado um vetor de coordenadas $x_i^{(k)}$ para cada um dos N pontos do meio. A etapa seguinte é utilizar essas informações para construir uma matriz de distâncias do tipo $(N \times N)$. Cada elemento da matriz representa a distância entre dois pontos quaisquer, através da métrica euclidiana de localização de pontos no espaço d dimensional. A matriz de distâncias, conserva na sua diagonal principal todos os elementos iguais a zero. Isto é, a distância de um ponto a ele mesmo é nula. Também é sabido que D é uma matriz simétrica em relação à diagonal principal.

$$D_{N\times N} = \begin{bmatrix} 0 & D_{12} & \cdots & D_{1N} \\ D_{21} & 0 & \cdots & D_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ D_{N1} & D_{N2} & \cdots & 0 \end{bmatrix}.$$

Em conformidade com a matriz de distâncias, acima representada, teremos um total de $N(N-1)/2$ distâncias aleatórias e simétricas $D_{ij} = D_{ji}$, sendo que, nem todas elas são variáveis independentes, devido à definição de espaço métrico, relação triangular de Schwartz (Eq. 2.2).

2.1.2 Condições Periódicas de Contorno

No problema da caminhada do turista, em razão da sua representação espacial, muitos pontos do mapa desordenado não se relacionam igualmente com os seus vizinhos. Os pontos localizados na área marginal do mapa, por exemplo, não dispõem da mesma região de vizinhança que os demais, arranjados na parte central do mesmo. Este padrão de isolamento na borda é denominado de efeito de borda, e se torna mais acentuado à medida que a dimensionalidade do meio se eleva.

O uso de condições periódicas de contorno (cpc), na estrutura geométrica do modelo, permite minimizar o efeito de borda na caminhada determinista do turista. Para sua aplicação é necessário unir as extremidades de cada uma das coordenadas espaciais, deformando o meio com condições aberta de contorno (cac), para um meio cuja geometria é circular em uma dimensão (Fig. 2.4), toróide (Fig. 2.5) em duas dimensões, e assim sucessivamente.

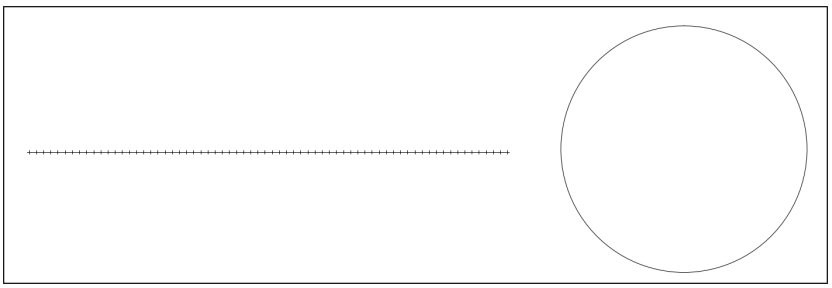


Figura 2.4: Condições abertas e periódicas de contorno em 1 dimensão.

2.1.3 O Algoritmo da Caminhada do Turista

Modelar fenômenos físicos através de métodos numéricos é uma prática muito usual entre os estudiosos, desde os primórdios, quando Von Neumann, popularizou essa idéia, na intenção

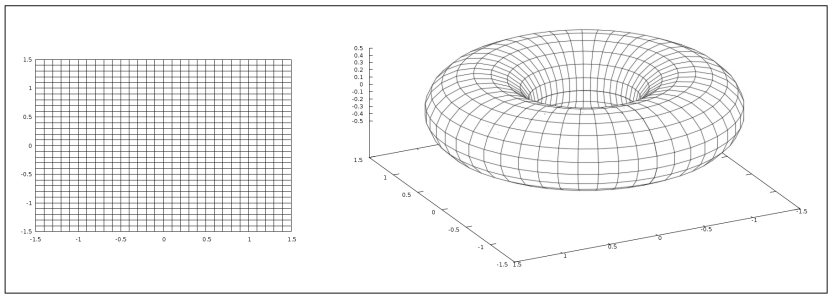


Figura 2.5: Condições abertas e periódicas de contorno em 2 dimensões.

de controlar o clima. Von Neumann construiu o seu próprio computador e, cercado de razões científicas, tinha a certeza de que a sociedade humana iria se libertar da desordem meteorológica. O seu trabalho, desde o início, era focado na teoria da turbulência hidrodinâmica. Complementando a isso, o pesquisador preveu que o experimento numérico era a grande solução para obter uma idéia do comportamento deste sistema dinâmico. Ele então imaginava que os cientistas calculariam as equações do movimento dos fluídos, e dessa forma realizariam previsões [3].

Neste caso, podemos afirmar que para obtermos mais informações do comportamento dinâmico na caminhada determinista do turista, devemos então utilizar ferramentas de modelagem numérica, afim de que essas auxiliem o nosso trabalho de investigação do conhecimento, em um menor dispêndio de tempo. Neste trabalho, o modelo de simulação numérico tomado como referência para o uso foi desenvolvido inicialmente por Gilson Francisco Lima, em [23] e uma descrição de partes do modelo podem ser conferidas nas próximas seções.

2.1.4 Matriz de Vizinhança

A dinâmica, a partir de cada ponto, no modelo determinista do turista, depende da localização dos $\mu+1$ vizinhos mais próximos, e não dos demais pontos que compõem o meio, ou seja, fora desse intervalo, a caminhada fica proibitiva em decorrência da regra estabelecida pelo modelo. Em outras palavras, o turista precisa conhecer somente a região próxima de onde ele se encontra. No entanto, apesar da localidade da regra, padrões globais complexos emergem desta caminhada.

Por esse motivo, constrói-se uma tabela de vizinhos, ordenada pelas distâncias euclidianas, (equação 2.2). Somente os $\mu+1$ vizinhos mais próximos de um ponto i qualquer são considerados. A medida que se considera todos os pontos gerados, tem-se uma matriz de dimensão $N\times(\mu+1)$. Cada elemento V_{ij} da matriz $V_{N\times\mu+1}$, aponta qual é o j-ésimo vizinho mais próximo do i-ésimo ponto.

$$V_{N\times\mu+1} = \begin{bmatrix} V_{1,1} & V_{1,2} & \cdots & V_{1,\mu+1} \\ V_{2,1} & V_{2,2} & \cdots & V_{2,\mu+1} \\ \vdots & \vdots & \ddots & \vdots \\ V_{N,1} & V_{N,2} & \cdots & V_{N,\mu+1} \end{bmatrix}.$$

No limite de alta dimensionalidade ($d \to \infty$), a correlação entre as distâncias tornam-se desprezíveis e as distâncias passam a ser consideradas como variáveis aleatórias independentes, devido ao enfraquecimento da desigualdade triangular[4]. Nesta situação temos o modelo de *ligações aleatórias (random link)* que conserva duas qualidades da métrica euclidiana:

- a distância de um ponto a si mesmo é igual a zero ($D_{ii} = 0, \forall i$) e

- a distância entre dois pontos (ou distância de ida e de volta) é sempre a mesma ($D_{ij} = D_{ji}, \forall ij$).

No entanto outros modelos de distâncias aleatórias podem ser imaginados. Por exemplo, o modelo de *mapeamento aleatório (random map)*, em que as distâncias são variáveis aleatórias, de modo que a distância entre dois pontos não necessitam imperativamente ser a mesma nos dois sentidos. No caso geral têm-se $D_{ij} \neq D_{ji}, \forall ij$. Também neste modelo, pode-se conservar a idéia da diagonal principal ser nula, que mantém zerada a distância de um ponto a ele mesmo.

Baseado na idéia do modelo de *mapeamento aleatório*, Derriba e Flyvbjerg [24], aplicaram uma variante no modelo, permitindo, neste caso, que a distância de um ponto a ele mesmo fosse também uma variável do tipo aleatória ($D_{ii} \neq 0$). Quando aplicado na caminhada determinista do turista, esse modelo não adquire as características de uma matriz aleatória simétrica, para tanto, basta gerar uma matriz de vizinhança, na qual, cada um dos μ vizinhos mais próximos são escolhidos aleatoriamente, com propriedades uniformes entre todos os N pontos. Isso é vantajoso para revelar novas propriedades dinâmicas no modelo.

Um resultado muito interessante demonstrado na Ref. [8], revela que a dinâmica no modelo determinista do turista pode desenvolver-se em qualquer matriz de vizinhança, desde que haja entre os elementos que integram o meio, uma ligação de ordem.

[4] A desigualdade triangular define que, em um triângulo, o comprimento de um dos lados é sempre inferior à soma dos comprimentos dos outros dois lados.

2.2 O Modelo da Caminhada do Turista como um Grafo Orientado e Ponderado

Um grafo pode ser compreendido como um conjunto V de vértices, ligados por U arestas. Cada ligação é associada a um par de vértices, dessa forma, um grafo pode ser esquematizado como $G = (V,U)$. Quando um grafo é dito orientado, a ligação entre um par de vértice, não é simétrica, ou seja, os arcos [5] são orientadas, de forma que se associa um sentido a cada ligação. Então, um grafo orientado é conectado somente por arcos. O modelo determinista do turista, pode ser compreendido como um grafo orientado, no qual as arestas representariam a relação entre os pontos (ou vértices) da tabela de vizinhança, através da dinâmica da caminhada.

Para recodarmos a construção do meio desordenado, d dimensional, começaremos fazendo uma alusão à distribuição aleatória, das coordenadas dos pontos $x_j^{(k)} = 1,2,...,N$, com densidade uniforme para cada uma das $k = 1,2,...,d$ coordenadas no intervalo entre $[0,1]$. Em seguida, calculam-se as possíveis distâncias entre cada par de pontos do meio, segundo as regras euclideanas[6] (ver a Seção 2.1.1). Estas distâncias são utilizadas para determinar os $\mu+1$ vizinhos mais próximos de um ponto x_i qualquer, construindo a tabela de vizinhança, que pode ser concebida como um grafo orientado. Os vértices são representados pelos pontos do mapa, e os arcos são as ligações entre os vizinhos mais próximos, os quais formam os caminhos, sendo indicados pela sucessão de arcos que se ligam. Os arcos são orientados na direção do deslocamento do caminhante, segundo a regra determinista do modelo.

O peso (ou grau) de um vértice é o número de arestas incidentes a ele, será denotado por k. Quando se trata de um grafo orientado, o grau de entrada é denotado por (k_i^{in}) e o grau de saída é denotado por (k_i^{out}), e se referem ao número de arcos que chegam e que saem de cada um dos vértices, respectivamente. No grafo orientado do turista, a quantidade de arcos emergentes de um dado vértice é igual ao número de elementos da tabela de vizinhança, $\mu+1$ dessa forma, a probabilidade k^{out} é fixa e é dada por:

$$P(k^{out}) = \delta_{k^{out},\mu+1} \ , \tag{2.3}$$

onde, $\delta_{i,j}$ é o delta de Kronecher definido por:

$$\delta_{i,j} = \begin{cases} 1 & \text{se} \quad i = j \\ 0 & \text{se} \quad i \neq j \end{cases} \ . \tag{2.4}$$

[5] Em um grafo, se as ligações $u_i : U, i = 1,2,...,m$, tem orientação, então estas são chamadas arcos. Se não tiverem orientação, então são chamadas arestas.

[6] Caso haja condição periódica de contorno, realiza-se o cálculo das distâncias relativas entre os pontos, de forma que se considere o encontro das extremidades do espaço geométrico.

Considere o grafo orientado do turista $G = (P,U)$, (Fig. 2.6) e a respectiva matriz de vizinhança (Tabela 2.1). Determinou-se que os arcos que receberam peso 0, detêm como primeiro vizinho mais próximo a si mesmo, os que receberam peso 1, como os segundos vizinhos mais próximos, os de peso 2, os terceiros vizinhos mais próximos, e assim sucessivamente, até o μ-ésimo vértice.

Veja-se que, para o grafo do turista com $\mu = 2$, de cada vértice (P_0 a P_4), saiam $\mu + 1$ arcos, indicando os três primeiros vizinhos mais próximos de cada um dos N pontos do espaço d dimensional. Como a regra do modelo determina, *deslocar para o ponto mais próximo que ainda não tenha sido visitado nos últimos μ intervalos de tempo*, então, é importante que se tenha uma tabela de vizinhança armazenando $\mu + 1$ pontos mais próximos de cada um dos N pontos, uma vez que essa é a forma do caminhante ter sempre um ponto para escapar, provendo um deslocamento mais extenso.

Acompanhando na tabela (Tabela 2.1), se o caminhante parte, por exemplo, do vértice P_1 para início da caminhada, ele se deslocará para o vértice P_0 que está à menor distância de P_1, depois de si[7]. Estando em P_0, o caminhante se desloca para o vértice P_2 (terceiro vizinho mais próximo), porque o mesmo não pode retornar para P_1 e nem permanecer onde se encontra, uma vez que tais vértices (pontos) se encontram na janela de memória μ. É dessa forma que o caminhante continua o seu deslocamento no grafo, até que entre em um ciclo e, a partir deste estado, o sistema deixa de evoluir.

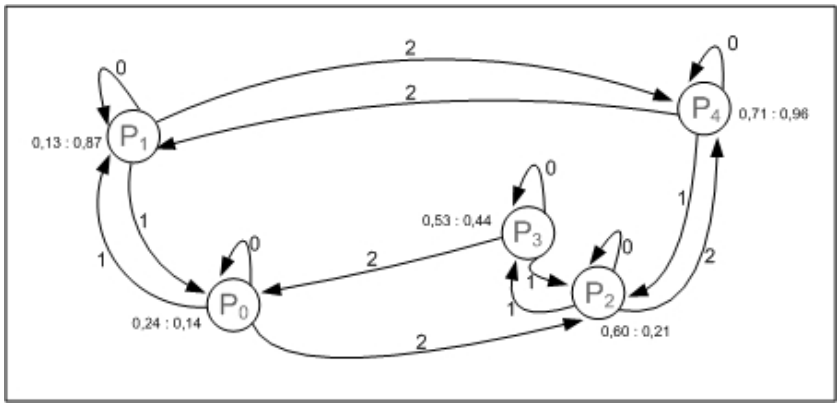


Figura 2.6: Exemplo de um grafo orientado e ponderado para o modelo determinista do turista com $\mu = 2$.

Devido à dispersão aleatória dos pontos no espaço d dimensional, o número de arcos que chegam em um dado vértice, não podem ser distribuídos de maneira uniforme, o que leva a

[7]Quando considera-se que o primeiro vizinho mais próximo de um ponto é ele mesmo, tem-se vértices do tipo anel, cujos arcos possuem as extremidades inicial e final coincidentes. Isso se aplica para o caso do turista preguiçoso ($\mu = 0$).

Pontos	Vizinhos		
	Peso 0	**Peso 1**	**Peso 2**
P_0	P_0	P_1	P_2
P_1	P_1	P_0	P_4
P_2	P_2	P_3	P_4
P_3	P_3	P_2	P_0
P_4	P_4	P_2	P_1

Tabela 2.1: Exemplo de tabela de vizinhança para o grafo orientado e ponderado do turista com $\mu = 2$.

prever apenas uma média[8] referente à probabilidde da ocorrência de k_i^{in}, e esta é definida por:

$$\langle k_{in} \rangle = E(k_{in}) = \mu + 1 \ . \tag{2.5}$$

2.3 Propriedades da Caminhada do Turista

De acordo com a memória μ empregada, as trajetórias geradas na caminhada determinista do turista, apresentarão algumas particularidades. Para determinados valores de μ, a caminhada produz comportamentos diferentes, no que diz respeito à formação de transientes e atratores. A memória impede que o sistema retorne a pontos recentementes visitados, originando assim situaçoes singulares. De acordo com essa característica, classificou-se a caminhada determinista do turista em três categorias: turista preguiçoso, turista sem memória, e turista com memória.

2.3.1 Turista Preguiçoso

O turista preguiçoso, é o turista que apresenta memória zero ($\mu = 0$). Neste caso, como havíamos visto na seção 2.1.4, a matriz de vizinhança, que detém na sua diagonal principal todos os elementos iguais a zero, impõe ao sistema uma única dinâmica, o ponto inicial, em todo intervalo de tempo é um atrator, com $p = 1$. A distribuição de atratores, neste caso é dada por $P_{0,d}(p) = \delta_{p,1}$. A diagonal principal igual a zero ($D_{ii} = 0$), denota que o ponto de partida possuirá sempre a menor distância dentre todos os outros N pontos do mapa, e o fato da memória ser igual a zero, permite que o caminhate permaneça onde está, apresentando um tempo de transiente sempre nulo[9], com distribuição igual a $P_{0,d}(t) = \delta_{t,0}$.

A situação $\mu = 0$ se torna interessante no modelo estocástico da caminhada do turista, onde os pontos (sítios) são visitados com uma dada probabilidade que depende da distância, entre-

[8] Medida formulada numericamente por Gilson F. Lima em [23], sob o uso de condições periódicas de contorno.
[9] Neste caso, o grafo do turista não apresenta arcos de ligação.

tanto, esse objeto de estudo não faz parte do escopo desta pesquisa. Outro fato curioso, está na aplicação da dinâmica do turista no modelo de campo médio, quando ($d \to \infty$), para o modelo de *mapeamento aleatório*, onde o turista preguiçoso pode não ocorrer, em decorrência da diagonal principal ser diferente de zero ($D_{ii} \neq 0$), o que implica em uma distribuição de ciclos [25].

2.3.2 Turista sem Memória

Outra particularidade do modelo determinista do turista é o caso do turista com memória um ($\mu = 1$). A cada intervalo de tempo, o caminhante traz à memória apenas o ponto em que se encontra, com isso, todos os outros pontos do meio, exceto o atual, se tornam propenso para visitação[10]. Porém, com esse valor de memória, o propósito da dinâmica é de que haja uma permutação, constante, entre dois pontos mutuamente próximos. Dessa forma, as trajetórias se concluirão em um par de pontos, também denominados de casais. Esses casais são os atratores do sistema, todos os ciclos formados em qualquer trajetória terão período 2, então a distribuição de períodos vale: $P_{1,d}(p) = \delta_{p,2}$. Este modelo pode ser resolvido analiticamente, ou seja, pode-se calcular a distribuição dos tempos de transiente. Este cálculo é mostrado nas referências [22, 26].

2.3.3 Turista com Memória

A proposta da caminhada determinista do turista com memória, como o próprio nome sugere, é considerar os valores de memória maiores ou iguais a dois ($\mu \geq 2$). Ao contrário dos casos para o turista preguiçoso e para o turista sem memória, a distribuição de atratores quando ($2 \leq \mu \leq N$) é pouco previsível, o menor período permitido para um atrator é $p = \mu + 1$. A dinâmica do sistema nesse caso, e a sua relação entre as grandezas da trajetória, desperta interesse, uma vez que pouco se conhece sobre a dinâmica do turista com memória. Ressalta-se que os primeiros estudos realizados se concentraram nos casos em que a memória $\mu \ll N$.

Dando sequência a essas descobertas, como indicam os trabalhos apresentados nas referências [23, 27, 26], o nosso objetivo é o de estudar a dinâmica da caminhada do turista, para o caso com memória, tendo como referência sistemas muito maiores do que os estudados anteriormente ($N \approx 10^4$). Chamamos a atenção para o fato de estarmos focados principalmente no caso em que a memória $\mu \sim N$.

[10] Neste caso, o grafo do turista apresenta em cada vértice, apenas uma ligação para o vértice de menor peso.

2.3.4 Conclusão

Neste capítulo, mostramos que a caminhada determinista do turista é um sistema dinâmico, cuja evolução se dá em um meio aleatório, tendo o tempo como uma variável discreta. Destacamos também o meio aleatório onde se desenvolve a dinâmica do modelo, bem como algumas particularidades do mesmo, como o efeito de borda, atenuado pelo uso das condições periódicas de contorno.

Também neste capítulo, apresentamos os aspéctos estáticos e dinâmicos do algoritmo da caminhada do turista, de forma que a dinâmica é mover-se progressivamente até que se fique aprisionado em pontos próximos, sobre uma matriz de vizinhança. Essa matriz é ordenada pela métrica euclidiana, a qual, se alteradas, permite examinar a dinâmica do turista em outros dois modelos: como os modelos de *ligações aleatórias* e *mapeamento aleatório*.

O estudo de determinadas informações estatísticas no modelo do turista, como as distribuições de tempos de transiente e de atrator foram primeiramente estudadas na referência [23]. Posteriormente, resultados analíticos envolvendo a caminhada do turista sem memória, no espaço unidemensional, foram abordados na referência [22]. Em vista desse desenvolvimento, resolvemos investigar a dinâmica do turista, explorando o seu comportamento com memória ($\mu \geq 2$), em dimensões maiores do que um.

3 *Nova Estrutura de Dados para o Modelo Numérico da Caminhada do Turista*

No meio científico, em áreas tão diferentes quanto a física, a química, a biologia, a economia, entre outras, geralmente existe a necessidade de se resolver problemas complexos, envolvendo grandes quantidades de dados. Esses dados precisam ser processados computacionalmente para posterior análise. Isso se dá pelo fato de que, nesse tipo de problema, não se interessa saber pelo todo simplesmente, sem simultaneamente compreender as interações das partes. Contudo, se as aplicações para esse tipo de interesse não forem bem planejadas, por exemplo, como o uso de recursos computacionais (disco, CPU[1], e memória RAM[2]), elas podem ocasionar prejuízos na busca pelas informações.

3.1 Caracterização do Problema

Em determinados experimentos numéricos, realizados na caminhada do turista com memória, observaram-se comportamentos curiosos, os quais não foram abordados nos primeiros estudos sobre o tema [23]. No uso do modelo determinista do turista, para sistemas muito grandes, o que implica uma geração significativa de dados, certos experimentos, obrigava o módulo de gerenciamento de memória RAM do sistema operacional GNU/Linux[3] a interferir no programa de simulação numérica, fazendo com que o mesmo finalizasse o processo[4] que estaria consumindo muito recurso computacional.

De acordo com a dinâmica do turista, cada ponto de partida produz trajetórias indepen-

[1] Central Processing Unit em inglês, ou Unidade Central de Processamento, também conhecida popularmente como microprocessador.

[2] Ao utilizar o termo RAM (Random Access Memory), estaremos nos referindo às memórias DRAM (Dynamic Random Access Memory).

[3] Tratado deste ponto em diante apenas por Linux

[4] Em sistemas operacionais, um processo é qualquer módulo executável, que disputa com outros módulos executáveis o acesso aos recursos computacionais partilhados.

dentes, compostas de uma parte transiente e de outra atrator. Percebeu-se que realizando o experimento com valores de memória μ muito alto, da ordem do sistema (N), em mapas unidimensionais, a caminhada gerava trajetórias complexas, ocasionando a formação de atratores de período p da ordem de N ou superiores, provocando um consumo muito grande de memória RAM do equipamento utilizado para o experimento. Esse fato, até então desconhecido, provocou enorme interesse no comportamento da dinâmica do turista, o que levou a investigar as razões dessa exploração total do meio de maneira tão inusitada.

3.1.1 O Experimento Numérico na Caminhada do Turista

O problema de algumas caminhadas serem finalizadas sem a sua completa conclusão atribuise ao fato de que, trajetórias que exploravam globalmente o sistema, geravam uma quantidade muito grande de dados. Dessa forma, a matriz, responsável pelo armazenamento dessas informações, fazia uso excessivo da RAM, exaurindo a memória. Para prosseguirmos com o trabalho, baseados neste foco, e obedecendo a regularidade de sistemas grandes ($N \approx 10^5$), determinadas modificações na estrutura de armazenamento das informações da caminhada fez-se necessária e estaraão sendo demonstradas nas próximas seções.

Para o modelo numérico do turista, considerou-se o programa desenvolvido por Gilson Franciso Lima, em seu trabalho de doutorado [23] e modificado, posteriormente, por Cesar Augusto Sangaletti Terçariol durante o mestrado [22]. O programa de simulação numérico sofreu várias implementações ao longo dos trabalhos de pesquisa. A primeira versão do programa foi feita em Fortran 77, sendo depois convertida para C ANSI[5].

A linguagem C foi escolhida como base de desenvolvimento do novo programa, pois tem uma promessa de uso genérico, é rápido e portável para diversas plataformas. Essa linguagem também nos oferece a vantagem de realizarmos as caminhadas deterministas em sistemas de maiores dimensões, por possuir funções capazes de manipular diretamente a RAM, seja através de tipos abstratos de dados[6] ou matriz. O programa de simulação numérico gera arquivos que contêm as informações necessárias para análise de tempo de processamento e consumo de memória, bem como todos os dados indispensáveis para que se faça interpretação estatística das trajetórias geradas pela caminhada do turista.

[5] Sigla do American National Standards Institute.

[6] Tipos abstratos de dados são estruturas especificamente construídas para armazenarem determinados tipos de dados. Estas estruturas especificam operadores que permitirem a manipulação destes dados, oferecendo recursos para que cada tipo possa ser processado junto com outros e até mesmo convertido em outro tipo.

3.1.1.1 A Estrutura de Armazenamento de Dados

Para que fique mais compreensível a metodologia de armazenamento dos dados no modelo numérico, discutiremos a seguir a respeito das análises das trajetórias. Ao final de cada caminhada, uma interpretação das trajetórias individualmente, permite determinar quanto tempo de transiente t, o caminhante andou até entrar em um atrator de período p. Posteriormente, uma descrição completa da caminhada se dá através da relação de transientes por períodos formados, gerando assim, a distribuição conjunta de transientes e de períodos $S^N_{\mu,d}(t,p)$. Essas informações são arranjadas em uma matriz bidimensional, de forma que as mesmas sejam dispostas de maneira contígua na estrutura.

Quando um caminhante fica preso em um atrator do tipo polígono complexo, Fig. 2.1(c), um resultado interessante é que o número de passos para dar a volta no atrator é maior do que o número de pontos que o compõe. Neste caso, podemos afirmar que temos um ponto de cruzamento dentro do atrator, característica exclusiva desse tipo de formação de período.

A importância desse resultado é que, de posse desses dados, estimamos com qual frequência um determinado tipo de atrator pode ser formado. Uma vez necessitados de representar mais uma grandeza da caminhada do turista, definimos a variável n_s como sendo a representação do conjunto de pontos que constitui um atrator de período p.

Como vimos anteriormente, os dados de tempo de transiente e os dados do tamanho do período, são representados em uma matriz do tipo $(t \times p)$, a quantidade de linhas e de colunas que delimitam essa matriz é definida pelo tamanho do maior transiente e do maior período formado em uma determinada trajetória. Para que possamos obter as informações do número de pontos que compõe cada um dos atratores, projetamos na estrutura de matriz bidimensional o índice n_s, e dessa forma, obtém-se uma matriz tridimensional $(t \times p \times n_s)$, que consiste respectivamente ao número de linhas, de colunas e de camadas da matriz.

A matriz, definida com mais uma dimensão, tornou-se muito grande e trouxe uma preocupação com relação à caminhada, pois a formação de períodos e transientes iguais a N, proporcionava em maior escala o desenvolvimento de matriz esparsa[7]. Nesse modelo de estrutura, poderia-se economizar espaços significativos de memória RAM se apenas os elementos diferentes de zero fossem armazenados. Além disso, as operações estatísticas sobre a matriz também seriam feitas em um tempo muito menor.

[7] Uma matriz é dita esparsa quando a maioria de seus elementos são iguais a zero.

3.1.1.2 Problemas na Estrutura do Tipo Matriz

Ao declarar uma matriz, o compilador alocará espaços de memória suficiente para conter o tamanho da estrutura, à medida que estes vão sendo solicitados. A quantidade real de memória RAM que será alocada vai depender do tipo de valor atribuído à matriz, e qual a plataforma computacional está sendo utilizada para o experimento. Para realizarmos uma análise do crescimento de uma matriz, examinamos um caso isolado, onde consideramos primeiramente que o espaço de memória RAM para um valor do tipo inteiro é de 4 bytes, (valor que pode ser obtido através da função *sizeof()*, linguagem C), com um arquitetura de 64 bits de instrução. Analisamos o consumo de memória RAM necessários para que uma matriz tridimensional atingisse o mínimo desfavorável para o computador.

Suponha que em uma caminhada, realizada no espaço bidimensional e com $N = O(10^4)$, obtiveram-se os respectivos resultados para o tempo de transiente, tamanho do período e n_s: $t = 40$, $p = 5000$ e $n_s = 3000$. Dos valores obtidos na caminhada, supomos que estes representam os maiores índices gerados. Para se presumir o quanto de memória RAM deverá ser disponibilizado para o armazenamento das informações da caminhada, multiplicaremos os valores do exemplo, pelo tamanho em bytes de uma variável do tipo inteiro, então, teremos um total de $2,4GB$ (gigabyte) de espaços alocados, que representa um gasto muito alto, que cresce proporcionalmente ao tamanho da matriz.

Destacamos que, para o exemplo anterior, não usamos o pior caso. Por esta razão, era indispensável desenvolver uma estrutura de armazenamento que se submetesse a manipular as informações na matriz de forma que os elementos pudessem ser alocados dinamicamente. Imaginemos uma estrutura cujos índices tivessem nomes abstratos dos valores correspondente, possibilitando assim, que estes não tenham relação com as posições de armazenamento do valor.

3.1.1.3 Tipos de Alocação de Dados para uma Estrutura de Armazenamento

A matriz que armazena as informações da caminhada do turista foi definida como uma matriz de alocação estática, sob o ponto de vista de armazenamento. De acordo com referência [28], uma matriz de alocação estática é aquela cuja vinculação de armazenamento ocorre quando a variável é declarada. Também na alocação estática, os elementos têm um tipo fixo de dado e são armazenados em posições consecutivas na memória. Por consequência, esse tipo de alocação, reduz a flexibilidade de manipulação dos elementos na matriz, dando possibilidades ao aparecimento de matrizes esparsas.

Do contrário, na alocação dos dados de maneira dinâmica, os elementos não precisam ter

um tamanho fixo e só podem ser alocados por meio de variáveis do tipo ponteiro ou variáveis do tipo referência, devido, especialmente, à imprevisibilidade do seu uso [28]. Na implementação desse paradigma, a estrutura de dados cresce apenas de maneira gradual ao tamanho do sistema. Nesse sentido, surge, então, apenas posições de memória que realmente terão seus espaços ocupados, com valores reais, à medida que vão sendo tomadas as informações da caminhada.

3.2 Lista Hierárquica -HL

O objeto lista de dados, foi a forma abstrata de se representar o armazenamento das informações no modelo numérico, proposto para a caminhada determinista do turista obter novos resultados, realizando experimentos com grandezas maiores. Uma lista é uma estrutura de dados linear e dinâmica, composta por nós que apontam para o próximo elemento, apenas o último nó não aponta para nenhum outro elemento da lista [28]. É habitual usar o tipo lista no armazenamento de dados, quando o número de elementos que irá compôr uma estrutura não é conhecido antecipadamente. Em uma lista, o primeiro elemento é denominado de "cabeça" da lista, e o último de "cauda". O elemento subsequente à "cabeça" é denominado de sucessor e o anterior à "cauda" é denominado de antecessor.

Projetar uma nova estrutura de armazenamento de dados para o modelo do turista incluiu pesquisar estruturas de dados que fossem adequadas às necessidades exigidas pela nova concepção de pesquisa no assunto, utilizando grandezas muito maiores do que era habitual, de maneira que isso não afetasse o uso excessivo de memória RAM. Por essa razão, a HL foi baseada nas características de três tipos de listas: generalizada, simplesmente encadeada e lista cruzada.

A lista generalizada tem como principal característica armazenar diferentes tipos de informações em uma única lista. A ideia da lista simplesmente encadeada é facilitar o acesso aos elementos da lista, enquanto que a lista cruzada previne o armazenamento de elementos nulos.

3.2.1 Estrutura Interna

A primeira lista na qual baseamos a estrutura interna do código é denominada lista generalizada. Uma lista L é dita generalizada quando uma sequência finita de $n \geq 0$ elementos, $\alpha_1, \ldots, \alpha_n$, onde α_i é ou um átomo ou uma sublista. Átomos são símbolos que identificam números e caracteres [28]. O elemento α_i, $1 \leq i \leq n$, que não são átomos, são ditos sublistas de L. Um exemplo de lista generalizada pode ser visualizado na Fig. 3.1.

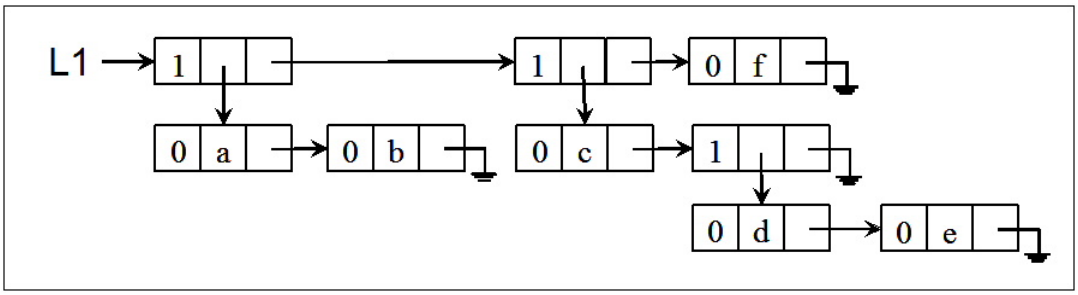


Figura 3.1: Exemplo de uma lista generalizada, onde mostramos que alguns elementos de uma lista pode ser uma sublista.

Nesse tipo de lista, o primeiro item de cada registro é denominado *tag* e indica se o conteúdo armazenado no item a frente é uma referência (ponteiro) para uma sublista, ou um átomo. Por átomo entende-se a informação a ser armazenada na lista e pode ser de tipo primitivo ou abstrato[8]. É comum utilizar um valor inteiro para representar o item *tag*, uma vez que as linguagens mais antigas não possuem um tipo booleano. Todavia, não há problema em utilizar variáveis booleanas quando disponíveis.

Na Fig. 3.1, caso o item *tag* contenha o valor "1", o próximo item é uma referência para uma sublista, caso contrário, se o item *tag* contiver o valor "0", o próximo item será um átomo. No exemplo apresentado, os átomos armazenados são variáveis do tipo caracter. Finalmente, o último item de cada registro é uma referência para o próximo elemento da lista principal. A essa referência dá-se o nome de CDR.

A segunda lista na qual nos baseamos é a lista simplesmente encadeada. Esta lista é a melhor maneira conhecida para representar sequências de dados lineares. Também é a melhor escolha, quando uma grande quantidade de operações do tipo inserção e remoção são necessárias em posições intermediárias. No entanto, não são destinadas a situações de acesso aleatórios, em que é necessário alterar entre elementos não sequenciais, uma vez que cada elemento de uma lista simplemente encadeada possuir uma referência para o próximo. Desta forma, para se obter um elemento da lista, é necessário percorrer todos os seus antecessores. A Fig. 3.2, exibe um exemplo desta lista, contendo quatro itens do tipo caracter, representados pelas letras de "a" até "d".

A terceira lista, utilizada para a implementação da estrutura HL, é a denominada lista cruzada. Esta lista é projetada para armazenar matrizes esparsas, uma vez que aqui os valores nulos não são armazenados. Este tipo de lista utiliza dois vetores para representar as linhas

[8]Tipos abstratos de dados são tipos definidos pelos programadores através de recursos disponibilizados pela linguagem de programação, como ponteiros, registros e uniões. Por outro lado, um tipo primitivo faz parte da linguagem. Como exemplo de tipos primitivos,disponíveis na linguagem C, podemos citar: int, float e char.

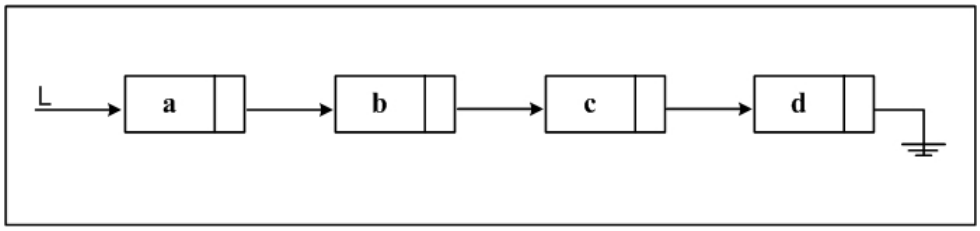


Figura 3.2: Exemplo de uma lista simplesmente encadeada, onde mostramos dados, do tipo caracter, armazenados de maneira sequencial.

e as colunas da matriz, respectivamente. Internamente, os elementos $a_{ij} \neq 0$ são armazenados em variáveis do tipo registro que podem ser acessados através de ponteiros. Assim, se uma matriz possui n_l linhas e n_c colunas, haverá n_l listas simplesmente encadeadas que representam as linhas e n_c para representar as colunas. Por isso, os elementos internos podem ser acessados, partindo tanto do vetor de linhas como do vetor de colunas. Na Fig. 3.3, é possível visualizar uma exemplo deste tipo de lista.

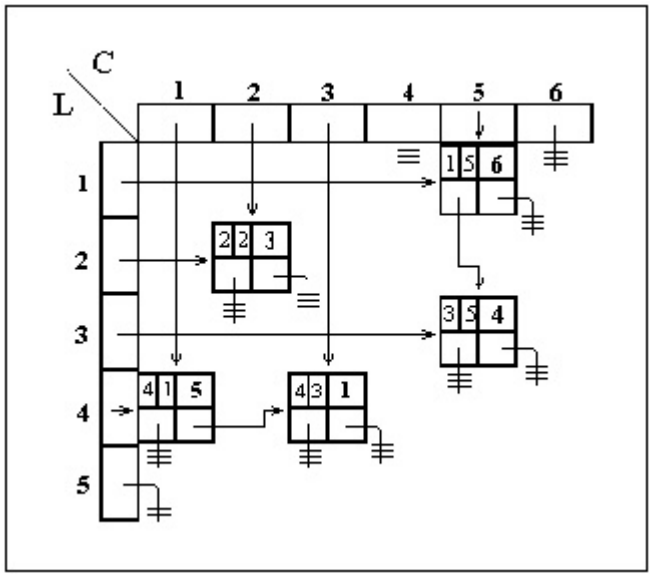


Figura 3.3: Exemplo de uma lista cruzada, onde matrizes são acessadas por listas de índices, linha e coluna.

Na Fig. 3.3, cada registro é composto de cinco itens, sendo que partindo da esquerda para a direita e de cima para baixo, temos:

- Linha: Identifica a linha em que o elemento está inserido. Essa variável faz-se necessária, pois nem todos os elementos de todas as linhas são armazenados. Assim, por exemplo, ao percorrermos a lista referenciada pela coluna “1”, obtemos como primeiro elemento o valor “5”, sendo assim, poderíamos supor que se trata de um item da primeira linha, uma vez que foi o primeiro a ser localizado. No entanto, como as três primeiras linhas

não possuem o primeiro elemento, estaríamos cometendo um erro, distorcendo a matriz esparsa original.

- Coluna: Identifica a coluna em que o elemento está inserido. A necessidade de sua representação segue a mesma argumentação do item Linha, descrito anteriormente.
- Elemento: representa o item armazenado, no caso da Fig. 3.3, itens do tipo inteiro.
- Referência para colunas: Armazena o endereço de memória do próximo item na mesma coluna da matriz.
- Referência para linhas: Armazena o endereço de memória do próxima item na mesma linha da matriz. armazenado na linha corrente, identificada pelo item Linha.

De fato, é possível que o acesso aos dados tornem-se mais lentos na estrutura proposta HL, se comparados com o tempo obtido por uma estrutura de armazenamento estático, como é o caso da matriz ($t \times p \times n_s$). Isso porque implementaram-se características de três tipos de lista, tornando a estrutura HL mais robusta. No entanto, quando realizamos experimentos com sistemas de ordem $N > 10^7$, o tempo de execução é menos importante se comparado ao consumo de memória RAM. No item 3.3.5, mostraremos que o tempo gasto pelo programa de simulação numérica com a estrutura proposta é praticamente o mesmo tempo obtido com o programa quando usa matriz estática, na estrutura de armazenamento dos dados.

3.2.2 Implementação da Estrutura de Dados

Iniciamos a implementação da nova estrutura de dados para cada caminhada do turista criando uma lista de armazenamento com três níveis. O primeiro nível compreende uma lista de transientes. Esta lista contém todas as informações sobre os transientes, obtidas nas caminhadas, bem como um ponteiro que referenciará diretamente o segundo nível da lista. O segundo nível da estrutura proposta é composto por todas as informações dos atratores de período p, gerados com as caminhadas, e de um ponteiro que apontará para o terceiro nível. E por último, no terceiro nível, são armazenadas as informações dos pontos que compõem um determinado período (n_s). Um modelo esquemático da estrutura HL pode ser visualizado na Fig. 3.4.

Como podemos observar, cada transiente possui uma lista de períodos, e cada período possui uma lista de pontos. Desta forma, quando um período ocorre em dois transientes distintos, eles precisam ser inseridos na lista de períodos de ambos. Esta regra também se aplica ao armazenamento de n_s. Para implementar a estrutura HL, fundamentados nas análises das trajetórias, definiu-se os seguintes campos de registro:

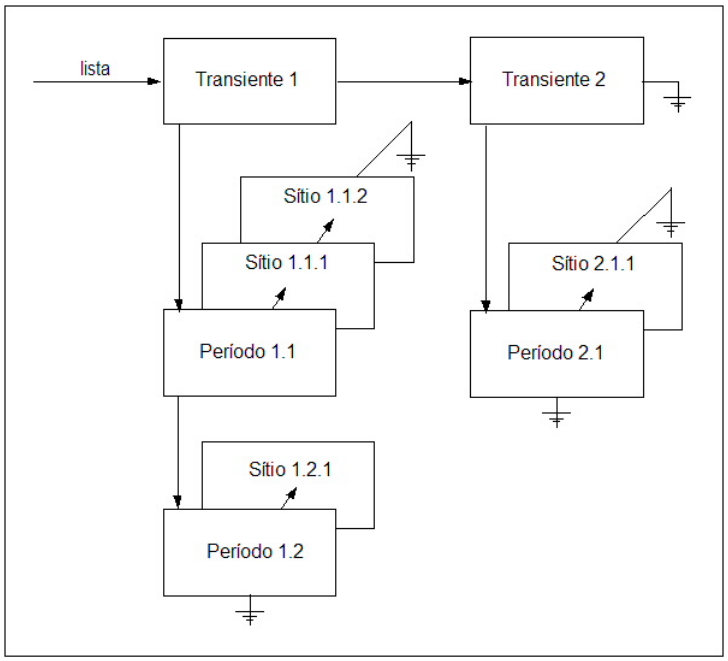


Figura 3.4: Modelo do estrutura HL desenvolvida.

- Registro TTransient: É utilizado para armazenar informações sobre os transientes obtidos nas trajetórias da caminhada. Este registro contém os campos: *trans*, armazena o tamanho do transiente e é utilizado como chave de acesso; *countTrans*, registra a frequência em que o tamanho do transiente, armazenado em trans, ocorreu durante as caminhadas; *next*, é um ponteiro para o próximo item na lista de transientes; LPeriod, é um ponteiro para a lista de períodos do transiente.

- Registro TPeriod: É utilizado para armazenar informações sobre os períodos, contendo os seguintes campos: *period*, utilizado para armazenar o período, sendo utilizado como chave de acesso; *countDist*, acumula a frequência em que $p = n_s$; *countRep*, acumula a frequência em que $p > n_s$; *next*, ponteiro para o próximo item da lista de períodos; *LPoint*, ponteiro para a lista de pontos (n_s).

- Registro TPoint: Armazena informações sobre os n_s, contendo os campos: *point*, armazena o n_s, sendo também utilizado como chave; *countPoint*, armazena a frequência em que n_s ocorre em um determinado período; *next*, é uma referência para o próximo item da lista de pontos.

Sempre que uma trajetória é finalizada, essas informações se tornarão disponíveis, desde que sejam inseridas na lista de maneira ordenada. Primeiramente os dados referentes ao transiente, na sequência os de período e, finalmente, os dados de n_s, mantendo a relação hierárquica presente na caminhada.

3.2.3 Obtendo as Distribuições Estatísticas na Lista Hierárquica

Para obter a distribuição marginal de transientes utilizando a estrutura HL, é necessário percorrer o primeiro nível da lista, coletando as informações contidas no campo *trans* e *countTrans*, uma vez que este totaliza a quantidade de vezes que um transiente ocorreu.

Para obter a distribuição marginal de períodos é necessário somar os campos *countDist* e *countRep*. Isto pode ser realizado criando um novo vetor dinâmico de períodos. Cada posição deste vetor armazenará a soma de um período p, onde p será considerado o índice desse vetor. Desta forma, o período, que ocorreu em mais de um transiente, terá suas informações acumuladas somente em uma posição do vetor dinâmico.

Esta abordagem torna-se interessante quando a distribuição conjunta de períodos e n_s não são necessárias. Caso contrário, deveríamos converter a estrutura HL em uma lista dinâmica de dois níveis, que conterá o período e n_s respectivamente, excluindo, assim, as informações de transiente. A Fig. 3.5 apresenta um esboço desta lista de dois níveis.

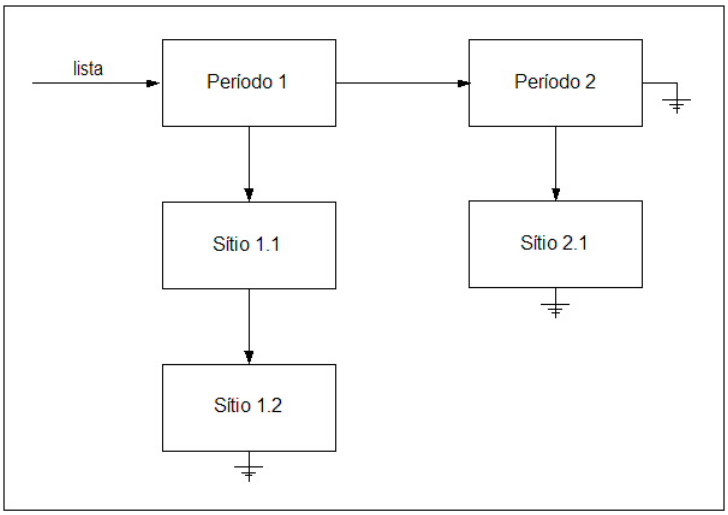


Figura 3.5: Esboço da lista de dois níveis utilizada para obter a distribuição conjunta entre período e n_s.

Desta forma, ao percorrer o primeiro nível da lista, temos a distribuição marginal de períodos. Já a distribuição conjunta de período e n_s pode ser obtida percorrendo as listas verticais para cada período armazenado no primeiro nível. Ao percorrê-las devem-se coletar as informações contidas nos campos *point* e *countPoint*.

A distribuição conjunta de transiente e período pode ser obtida na mesma direção que a distribuição conjunta de períodos e n_s. Desta forma, é necessário percorrer a lista de transientes e, para cada nó visitado, percorrer a lista de períodos (no nível imediatamente abaixo) obtendo informações dos campos *period* e a soma dos campos *countDist* e *countRep*.

3.3 Resultados

Para se confirmar a eficiência da estrutura HL, realizamos experimentos numéricos utilizando o modelo numérico atual, que apresenta um armazenamento estático dos dados, e o modelo modificado, com as características dinâmicas de armazenamento propostas. Fixou-se um número de realizações para os experimentos, e relacionaram-se as demais grandezas de interesse:

- dimensão (d): 1, 2 e 3;
- número de sítios (N): 1000, 5000, 10000, 25000 e 50000;
- memória (μ): 5, 20 e 50.

O objetivo final é comparar o consumo de memória RAM entre os modelos, bem como o tempo de processamento exigido por cada realização.

3.3.1 Metodologia

Considerou-se uma confiança de 95%, (equação 3.1), sendo $T_{\alpha/2}$ a confiança obtida através da tabela *T-Student*, com $M-1$ graus de liberdade [29]. M representa o número de realizações, enquanto S representa o desvio-padrão. Para este trabalho foram considerados $M = 10$ experimentos numéricos.

$$\overline{X} - T_{\alpha/2} \times \frac{S}{\sqrt{M}} \leq X \leq \overline{X} + T_{\alpha/2} \times \frac{S}{\sqrt{M}} \tag{3.1}$$

O computador usado para executar as simulações numéricas é composto por um processador Intel Xeon 3GHz 64 bits de núcleo duplo com 4 MB de cache L2 e 2GB de memória RAM. Sistma operacional Linux, com *kernel* 2.6.18, compilado para arquitetura X86_64, mais o compilador gcc, versão 4.1.2.

Para obtermos o valor consumido de memória RAM, pelo modelo numérico atual, considerou-se a equação 3.2. As respectivas grandezas t, p e n_s representam as dimensões da matriz. Estas armazenarão os maiores valores obtidos por essas grandezas, no final de cada caminhada. A função *sizeof()*, retorna a quantidade de memória RAM, consumida em *bytes*.

$$\text{Memória Consumida} = t \times p \times n_s \times sizeof(long) \tag{3.2}$$

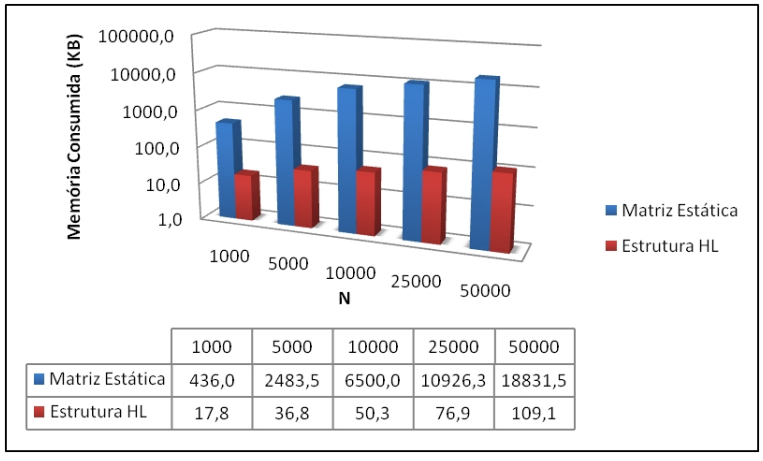
Para obtermos o valor consumido de memória RAM, pelo modelo numérico proposto, considerou-se a equação 3.3. A grandeza tt representa a quantidade de registros (TTransient), armazenados na estrutura, assim como tp e ts representam, respectivamente, a quantidade de registros (TPeriod) e (TPoint).

$$\begin{aligned} \text{Memória Consumida} = \quad & tt \quad \times sizeof(TTransient) \\ & + \quad tp \times sizeof(TPeriod) \\ & + \quad ts \times sizeof(TPoint) \end{aligned} \tag{3.3}$$

Nas próximas seções, estão apresentados os gráficos comparativos de ambos modelos de armazenamento. A ideia de criar uma estrutura de armazenamento, baseada em níveis de listas de dados, proporcionou o baixo consumo de RAM, uma vez que os dados só são armazenados à medida que forem necessários, ou seja, de acordo com a demanda de informações geradas.

3.3.2 Uma Dimensão

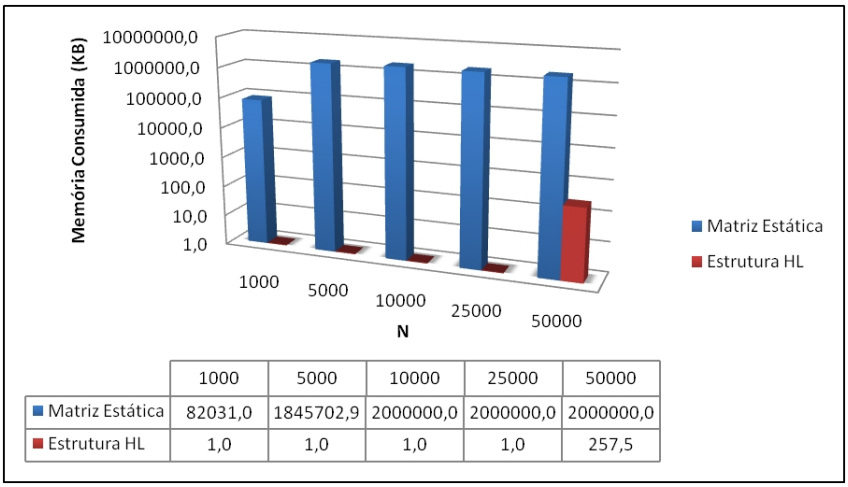
A Fig. 3.6 apresenta a utilização de memória obtida nos experimentos numéricos para ambas estruturas de dados apresentadas anteriormente. Para tal, usamos como valor de memória $\mu = 5$.



	1000	5000	10000	25000	50000
Matriz Estática	436,0	2483,5	6500,0	10926,3	18831,5
Estrutura HL	17,8	36,8	50,3	76,9	109,1

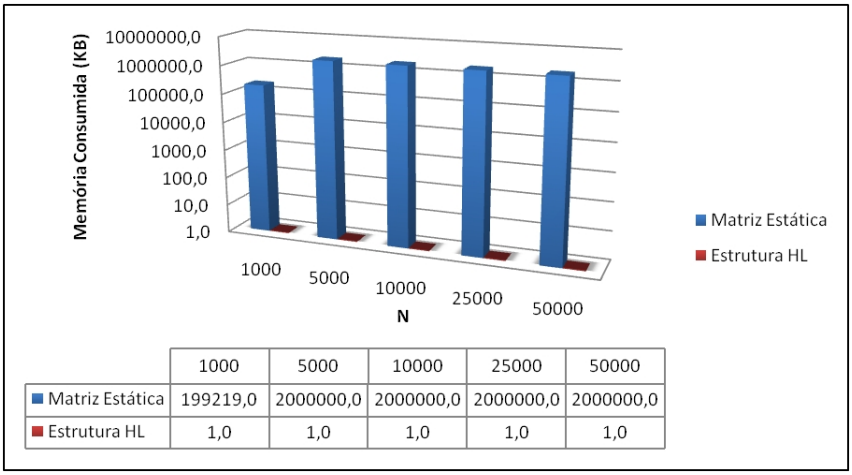
Figura 3.6: Consumo de memória RAM no modelo do turista para $\mu = 5$.

Ressalta-se que nos gráficos apresentados nas seções 3.3.2, 3.3.3 e 3.3.4, consideramos que quando o processo de simulação numérica é abortado pelo sistema operacional, associa-se o valor $2GB$, tido como o valor total de memória disponível na máquina utilizada para realização do experimento. Outra observação importante refere-se a escala do eixo y. Utilizamos a escala



	1000	5000	10000	25000	50000
Matriz Estática	82031,0	1845702,9	2000000,0	2000000,0	2000000,0
Estrutura HL	1,0	1,0	1,0	1,0	257,5

Figura 3.7: Consumo de memória RAM no modelo do turista para $\mu = 20$.



	1000	5000	10000	25000	50000
Matriz Estática	199219,0	2000000,0	2000000,0	2000000,0	2000000,0
Estrutura HL	1,0	1,0	1,0	1,0	1,0

Figura 3.8: Consumo de memória RAM no modelo do turista para $\mu = 50$.

logarítmica para este eixo, uma vez que, em alguns casos, a diferença entre a quantidade de memória consumida pelas estruturas analisadas é muito elevada, dificultando a visualização dos resultados. O valor máximo foi determinado pelo maior valor de memória consumida, obtida no processo de simulação. Por esse motivo, os gráficos gerados podem ser diferenciados na escala do eixo y. Também para esses dados, é importante observar que não houveram sobreposições dos intervalos de confiança dos resultados obtidos nos experimentos. Desta forma, optou-se por omitir essa informação dos gráficos, flexibilizando a construção dos gráficos.

A partir do gráfico da Fig. 3.6, e pelos resultados obtidos nos experimentos numéricos, observamos que a matriz estática utiliza aproximadamente 18 MB de memória para $N = 50000$ e 436 KB para $N = 1000$. Estes são os valores máximo e mínimo de memória consumida pela matriz, enquanto a estrutura HL consumiu aproximadamente 109 KB e 17 KB, respectivamente, para os mesmos valores de N. Desta forma, é possível afirmar que a estrutura de dados HL consome apenas 0,58% e 4% da memória utilizada pela matriz estática para ambos os valores

de N.

Quando o valor de memória μ sobe para 20 (Fig. 3.7), a matriz estática executa apenas o experimento em que $N \leq 5000$. Para este valor de N, a memória consumida foi de aproximadamente 1,9 GB. Para este cenário, a estrutura HL utilizou aproximadamente 1 KB para $N \leq 15000$. Em decorrência do consumo excessivo da memória RAM, que se dá pelo armazenamento dos dados através da matriz estática, o processo de simulação numérica é abortado pelo sistema operacional. Isso ocorreu porque o experimento ocasionou um estado de *trashing*[9].

O valor máximo de memória consumida pela estrutura HL, com $\mu = 20$, foi 257 KB com $N = 50000$. É possível perceber que com os valores $N \leq 25000$ e $\mu = 20$, a HL consome menos memória que o apresentado com a memória $\mu = 5$. Isto ocorre devido à presença da memória crítica encontrada na caminhada do turista unidimensional [27]. Verificou-se que, a partir de um determinado valor de memória, todas as caminhadas executadas formam trajetórias capazes de percolarem o sistema[10]. Tal fenômeno acontece para qualquer valor de N, sendo caracterizados pela formação de períodos do tipo $p = N$, convergindo para uma região em que $p > N$. Esse comportamento será abordado no capítulo 4.

Ao considerarmos $\mu = 5$, percebe-se que tal valor é crítico apenas para mapas com $N \leq 33$[11]. Assim, este valor de μ não é critico para nenhum valor de N incluido no planejamento de experimento, o que leva a formação de atratores com diferentes formatos e períodos, consumindo mais de uma posição na estrututa HL. Ao analizarmos $\mu = 20$, percebe-se que tal valor é crítico para cenários com $N \leq 10^6$, fazendo com que todas as caminhadas percolem o sistema, gerando períodos do tipo $p = N$.

Tal fato permite a utilização de apenas uma posição em cada uma das estrutura HL, consumindo apenas $1KB$ de memória RAM. Do contrário, a matriz estática alocará N posições para períodos e n_s, consumindo muita memória RAM. Já para $N = 50000$, percebe-se que algumas trajetórias geradas estão na zona de convergência para o valor de μ estudado, elevando o consumo de memória para $257,5KB$ aproximadamente.

Quando aumentamos o valor de μ para 50, o sistema operacional abortou o processo de simulação, utilizando a matriz estática, com $N > 1000$. Para valores inferiores, a memória consumida é aproximadamente 200 MB, enquanto que com a estrutura HL utilizou-se 1 KB para todos os valores de N testados. Estes resultados podem ser conferidos na Fig. 3.8

[9] Diz-se que um processo está em trashing, quando ele passa a maior parte do seu tempo de execução no processo de paginação (realocação de memória, virtual), implicando na baixa utilização da CPU.

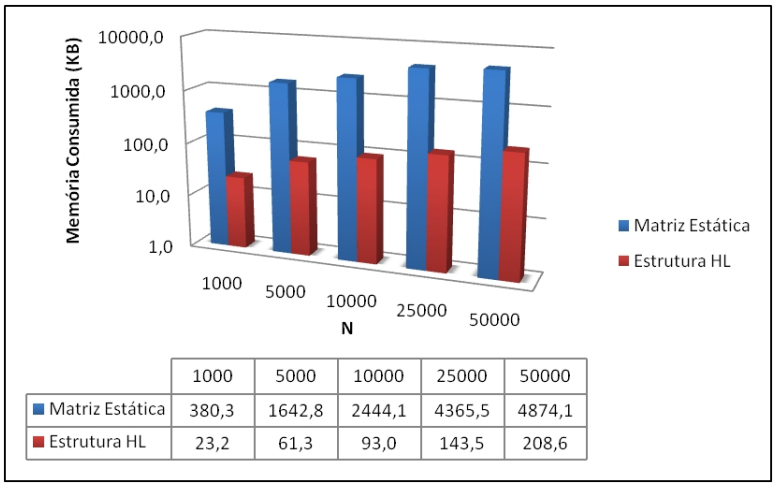
[10] Considera-se que a trajetória em uma dimensão percolou quando todos os pontos foram visitados, de forma que o caminhante atravessou o sistema de uma extremidade à outra.

[11] Um valor crítico de memória é aquele que permite que o caminhante explore ao máximo o ambiente, produzindo trajetórias com comprimento de ordem N.

Com a melhoria no algoritmo, pudemos perceber novos efeitos na caminhada do turista com memória grande. Estas novidades serão apresentadas e discutidas nos capítulos 3 e 4.

3.3.3 Duas Dimensões

O fato da caminhada do turista bidimensional não apresentar muitas trajetórias que geram períodos da ordem de N, possibilitou que o algoritmo desenvolvido com a matriz estática, realizasse todos os experimentos, onde os respectivos valores de memória 5, 20 e 50, foram utilizados em sistemas com $N = 1000, 5000, 10000, 25000$ e 50000. Os resultados obtidos com os dados exibidos acima, também podem ser observados, através dos gráficos esquematizados nas próximas figuras.

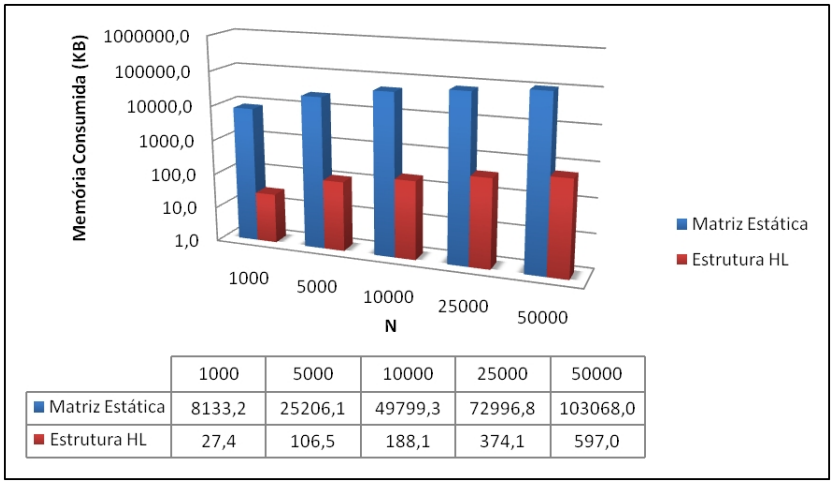


	1000	5000	10000	25000	50000
Matriz Estática	380,3	1642,8	2444,1	4365,5	4874,1
Estrutura HL	23,2	61,3	93,0	143,5	208,6

Figura 3.9: Consumo de memória RAM no modelo do turista para $\mu = 5$.

Na Fig. 3.9, a matriz estática consumiu 380 KB e 4.365 KB para $N = 1000$ e $N = 25000$, respectivamente. Esses são os valores mínimo e máximo de consumo de memória apresentados no experimento, enquanto a estrutura HL consumiu 23 KB e 143 KB para os mesmos valores de N. Desta forma, podemos afirmar que a estrutura HL consome, aproximadamente, apenas 6% e 3% da quantidade de memória alocada pela matriz estática, para ambos os valores citados de N. Quando analisamos a diferença de memória consumida, observamos que a estrutura HL consome até, aproximadamente, 4,6 MB a menos de memória que a matriz estática. Tal fato foi registrado com $N = 50000$, cujo consumo de memória na matriz estática foi de 4.874 KB.

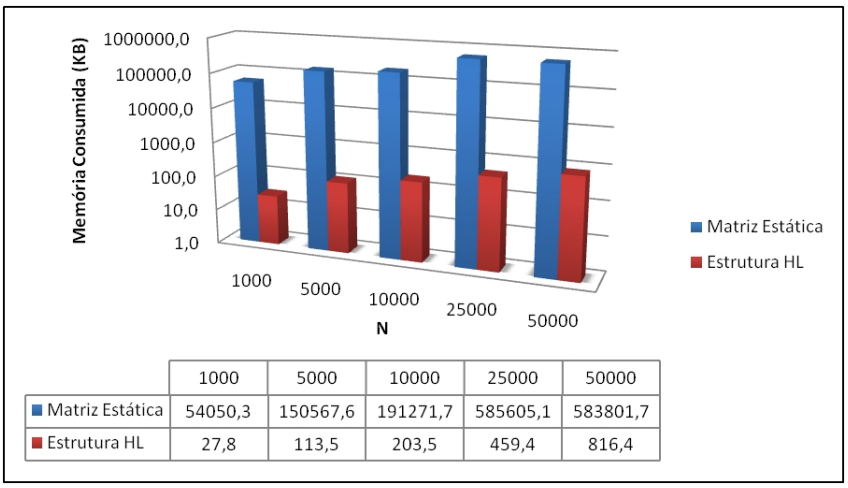
Quando realizamos os experimentos com $\mu = 20$, a estrutura HL apresentou-se ainda mais eficiente. A diferença de memória consumida foi de 102 MB, ao considerar $N = 50000$. A redução média do consumo foi de 99.55%, como pode ser visualizado na Fig. 3.10. O valor máximo de memória consumida foi de 597 KB pela estrutura HL e aproximadamente 103 MB pela matriz estática. Este fato reforça a eficiência da estrutura HL aqui apresentada.



	1000	5000	10000	25000	50000
Matriz Estática	8133,2	25206,1	49799,3	72996,8	103068,0
Estrutura HL	27,4	106,5	188,1	374,1	597,0

Figura 3.10: Consumo de memória RAM no modelo do turista para $\mu = 20$.

Também para $\mu = 50$, o gráfico da Fig. 3.11 demonstra que redução média do consumo de memória foi de 99.91% aproximadamente. O consumo máximo registrado foi de aproximadamente 585 MB, na matriz estática, e 459 KB na estrutura HL. Isto representa uma diferença de $O(10^3)$.

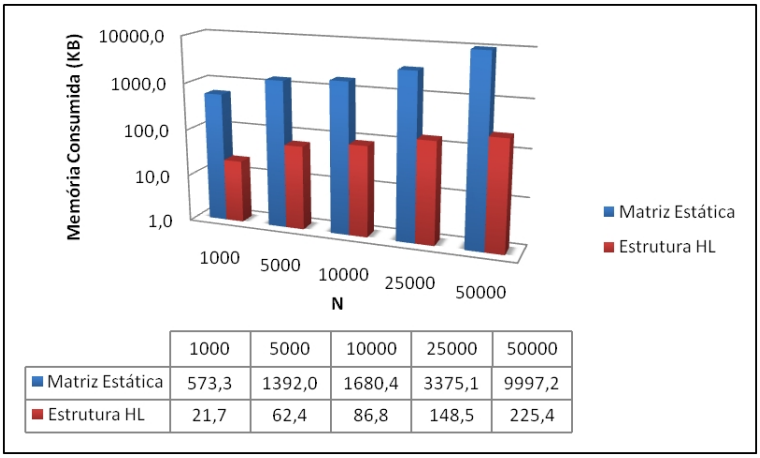


	1000	5000	10000	25000	50000
Matriz Estática	54050,3	150567,6	191271,7	585605,1	583801,7
Estrutura HL	27,8	113,5	203,5	459,4	816,4

Figura 3.11: Consumo de memória RAM no modelo do turista para $\mu = 50$.

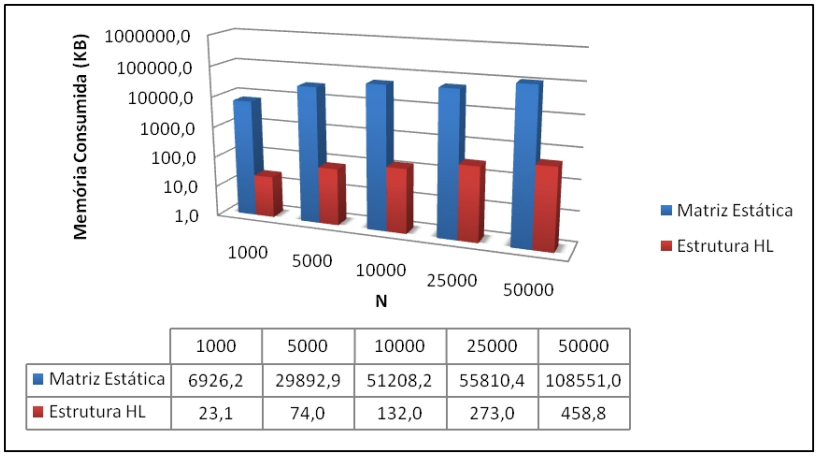
3.3.4 Três Dimensões

Quando simulamos a caminhada do turista em um espaço tridimensional, observou-se que a estrutura HL obteve a maior vantagem registrada sobre a matriz estática, se comparadas com os resultados obtidos nas dimensões um e dois. Isto foi possível por que existe uma probabilidade maior de combinações dos fatores: transientes, períodos e n_s, uma vez que o número de possibilidades de deslocamento para o turista, em uma dado ponto, é maior neste caso do que em dimensões menores, como é ilustrado nas Fig. 3.12, 3.13 e 3.14.



	1000	5000	10000	25000	50000
Matriz Estática	573,3	1392,0	1680,4	3375,1	9997,2
Estrutura HL	21,7	62,4	86,8	148,5	225,4

Figura 3.12: Consumo de memória RAM no modelo do turista para $\mu = 5$.

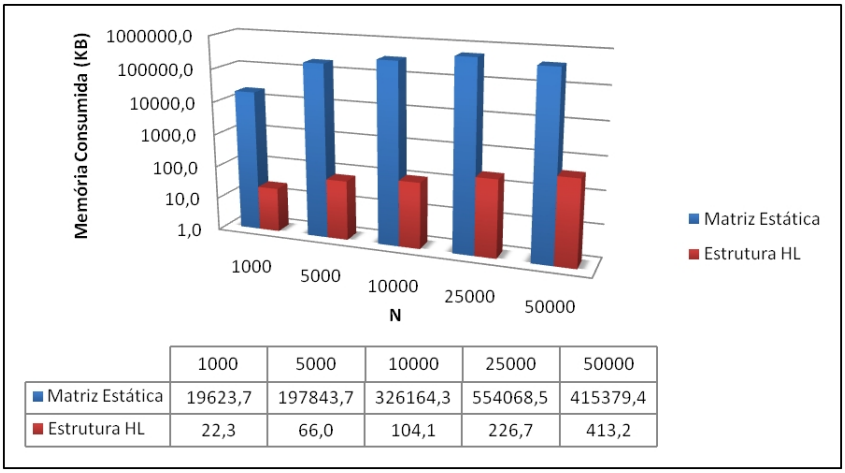


	1000	5000	10000	25000	50000
Matriz Estática	6926,2	29892,9	51208,2	55810,4	108551,0
Estrutura HL	23,1	74,0	132,0	273,0	458,8

Figura 3.13: Consumo de memória RAM no modelo do turista para $\mu = 20$.

Analisando o gráfico da Fig. 3.12, observamos que o pior resultado da matriz estática ocorreu quando o experimento empregou o maior valor N. Foram consumidos aproximadamente 9 MB de memória RAM, enquanto o pior resultado da estrutura HL apresentou um consumo de apenas 225 KB. Isto representa uma redução de 97,75% no consumo de memória, obitido pela estrutura proposta. Por outro lado, o melhor resultado obtido pela matriz estática, para o valor de μ considerado, foi 573 KB quando $N = 1000$. Para este valor de N, a estrutura HL consome apenas 21 KB de memória RAM. Isto representa um consumo 27 vezes menor do que o apresentado na matriz estática.

Para o valor de memória $\mu = 20$ (Fig. 3.13), a redução média do consumo de memória da estrutura HL é de 99,65%. A menor redução registrada foi de 99.51%, quando a matriz estática consumia aproximadamente 55 MB, a estrutura HL consumiu apenas 273 KB. Este resultado foi obtido com $N = 25000$. O maior ganho foi registrado para $N = 5000$, quando a matriz estática utilizou aproximadamente 30 MB de memória e a estrutura HL consumiu somente 74



	1000	5000	10000	25000	50000
Matriz Estática	19623,7	197843,7	326164,3	554068,5	415379,4
Estrutura HL	22,3	66,0	104,1	226,7	413,2

Figura 3.14: Consumo de memória RAM no modelo do turista para $\mu = 50$.

KB.

Baseado na explicação anterior, entendemos que o melhor resultado obtido pela estrutura HL sobre a matriz estática ocorreu com $N = 5000$, enquanto o pior ocorreu com $N = 25000$. Dessa forma, quando aumentamos o valor de N, as chances de diferentes combinações para transiente, período e n_s também aumentam. Isso faz com que a estrutura HL aloque mais espaços de memória, enquanto a quantidade máxima de memória consumida pela estrutura estática é determinada unicamente pelo tamanho do maior transiente, do maior período e do maior valor de n_s formados.

Quando $\mu = 50$ (Fig. 3.14), a matriz estática consome 554 MB de memória para $N = 25000$ no pior caso e, a estrutura HL, também no pior caso, consome 413 KB com $N = 50000$. Para este caso, a redução média, ao se utilizar a estrutura HL, foi de 99,94%. Já a maior redução no consumo de memória ocorreu para $N = 10000$ com um valor de 99,96%.

3.3.5 Análise de Tempo de Processamento

Como havíamos mencionado na seção 3.1.1.2, o objetivo da estrutura de dados proposta é a economia de memória RAM, porém, o tempo consumido pelo *software* de simulação, ao utilizar a estrutura HL, é um fator importante para sua adoção. Baseado nesta informação, analisamos, também, o tempo consumido pelo simulador ao utilizar ambas as estruturas de dados. Para tal, foram considerados os mesmos fatores e níveis do planejamento de experimento desenvolvido para a análise do consumo de memória.

3.3.5.1 Espaço Unidimensional

A Fig. 3.15 apresenta a razão entre o tempo consumido pelo *software* de simulação ao utilizar a estrutura HL e a matriz estática em um espaço unidimensional. Dessa forma, se o valor obtido for menor do que um, o tempo gasto pelo simulador ao utilizar a estrutura de dados HL foi menor, por outro lado, se o valor apresentado no gráfico for maior que um, a utilização da matriz estática demandou menos tempo para executar os experimentos. Finalmente, se a razão obtida for um, o tempo consumido pelo programa de simulação foi similar para ambas as estruturas. Uma informação que não se deve esquecer é que, nos casos em que algum experimento foi abortado, o gráfico não apresentou a razão do tempo consumido para determinados valores de N.

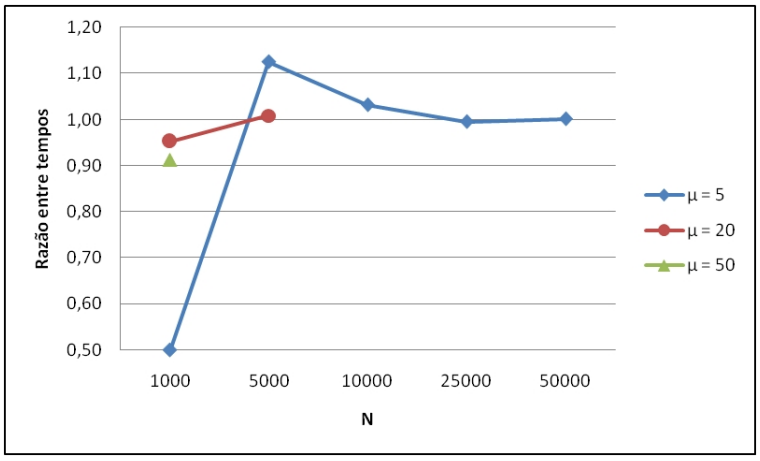


Figura 3.15: Razão do tempo de simulação consumido pelo experimento ao utilizar a estrutura HL e a matriz estática, considerando diferentes valores de μ.

Analisando o gráfico apresentado na Fig. 3.15, entendemos que o tempo consumido pela simulação, ao utilizar a estrutura HL, foi menor para pequenos valores de N e $\mu = 5$. Já para $5000 \leq N \leq 10000$, as simulações que utilizaram a matriz estática consumiram menos tempo. Todavia, com $N \geq 25000$, os tempos consumidos, ao utilizar ambas as estruturas, foram semelhantes.

A diferença foi significante apenas para os primeiros dois valores de N, sendo que a utilização da estrutura HL consumiu aproximadamente 50% menos tempo para $N = 1000$. Por outro lado, para $N = 5000$, a matriz estática consumiu 11% menos memória.

O mesmo comportamento ocorreu quando aumentamos o valor de μ para 20 ou 50. O programa de simulação consome menos para pequenos valores de N, ao utilizar a estrutura HL. Neste cenário não foi possível analisar o tempo consumido para $N \geq 5000$, uma vez que a utilização da matriz estática levou o sistema operacional ao estado de *trashing*.

3.3.5.2 Espaço Bidimensional

O tempo de simulação consumido no espaço bidimensional pode ser visto na Fig. 3.16. Nesta dimensão, as razões obtidas entre os tempos consumidos são diferentes daqueles apresentados na dimensão um.

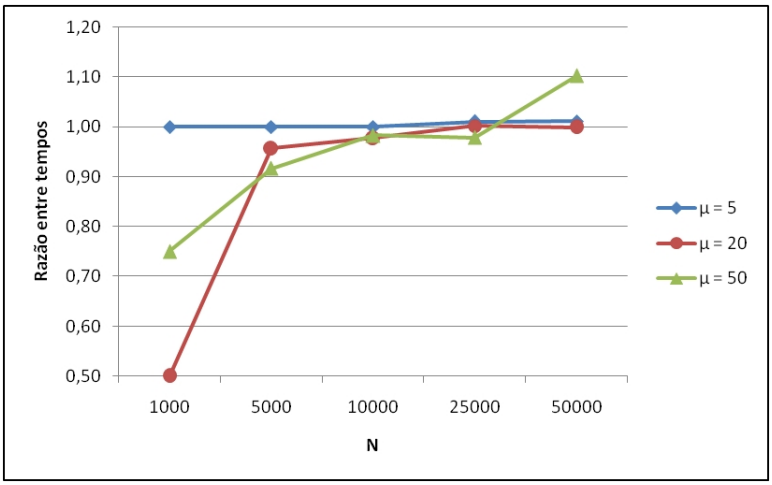


Figura 3.16: Razão do tempo de simulação consumido pelo experimento, ao utilizar a estrutura HL e a matriz estática, considerando diferentes valores de μ.

Podemos observar que para $\mu = 5$, o tempo consumido ao utilizar ambas as estruturas de dados é semelhante, porém, quando aumentamos o valor de μ, a estrutura HL passa a consumir menos tempo na maioria dos casos. Isto é possível por que o tamanho da matriz estática não é conhecido de início. Assim, é necessário realocar o espaço de memória quando novos valores de transientes, de períodos ou de n_s são gerados, desde que eles sejam maiores que os valores alocados para as dimensões $t \times p \times n_s$.

Na Fig. 3.16 é possível concluir que a estrutura HL é mais eficiente para pequenos valores de N. Por exemplo, com $N = 1000$, o tempo gasto ao utilizar a estrutura HL foi 50% menor ao considerar $\mu = 20$. Já para $\mu = 50$ a vantagem foi de 25%. Quando o valor de μ aumenta, o tempo consumido por ambas as estruturas é similar.

Ao comparar os resultados obtidos em uma e duas dimensões, concluímos que para $\mu = 5$ e $N \geq 5000$, a diferença entre o tempo consumido por ambas as estruturas é insignificante. Para $\mu \geq 20$, esta comparação não pode ser realizada, devido ao estado de *trashing*, relatado anteriormente.

3.3.5.3 Espaço Tridimensional

Nesta dimensão, a maior vantagem obtida com a utilização da matriz estática ocorreu com o uso dos menores parâmetros de memória e do tamanho do sistema (ver Fig. 3.17). Com esta configuração, o tempo consumido por ela foi 15% menos que o consumido pela estrutura HL. Por outro lado, a estrutura HL registrou ganhos com $\mu = 50$ tanto para $N = 1000$, quanto para $N = 5000$, cuja redução de tempo foi de 33% e 7%, respectivamente. Para $\mu = 20$ ambas as estruturas de dados apresentaram desempenhos semelhantes, com pequenas vantagens registradas ora para a estrutura HL, ora para a matriz estática, todavia, tais diferenças são muito pouco significantes.

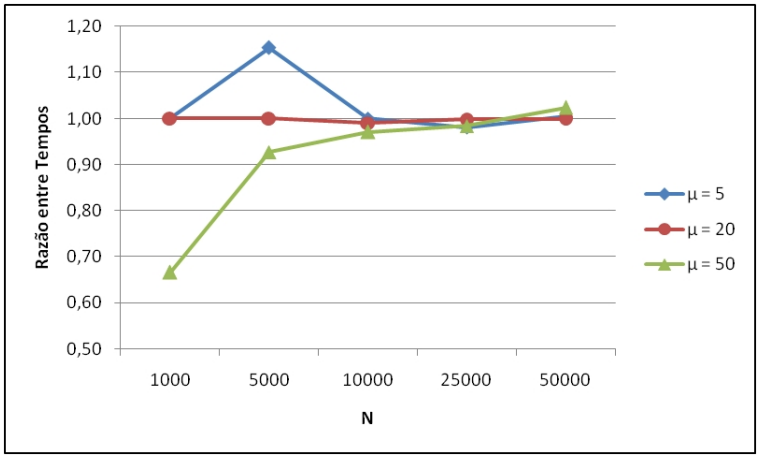


Figura 3.17: Razão do tempo de simulação consumido pelo experimento, ao utilizar a estrutura HL e a matriz estática, considerando diferentes valores de μ.

3.3.6 Conclusão

Neste capítulo apresentamos uma nova estrutura de armazenamento de dados, para o modelo numérico do turista determinista. A estrutura proposta foi criada com a ideia de se aplicar ao modelo numérico, grandezas consideradas de ordem alta, sem provocar o uso excessivo de recurso computacional, como o uso da memória RAM. Por essa razão, o modelo atual, que dispõe do uso da matriz estática, restringiu estudos na caminhada à pequenos valores de μ e N. Através do modelo numérico modificado, pode-se analisar diferentes configurações da caminhada, bem como analisar mais uma variável de interesse, (n_s), que armazena todos os pontos que constituem um atrator, permitindo a descoberta de novos comportamentos no sistema. Do mesmo modo, a estrutura HL, proporcionou a saída da informação conjunta, de tempos de transiente e de períodos do atrator.

Mostrou-se os métodos de fácil aplicação, para obter as distribuições estatísticas com essa

nova estrutura. Também introduzimos com a alocação dinâmica de memória, uma nova maneira de abordar o problema numérico das distribuições conjuntas de tempos de transiente e período do atrator. Para obter as distribuições conjuntas, precisa-se gerar uma outra matriz dinâmica, obtida facilmente da estrutura HL. As distribuições marginais de transiente e de período, podem ser obtidas simplesmente percorrendo a estrutura que foi proposta.

Finalmente, comprovou-se a eficiência da estrutura HL, mostrando que esta consome menos memória RAM que a matriz estática. Nos experimentos, registramos em média, economias de memória de 99,49%, 98,41% e 98,52% para as dimensões um, dois e três, respectivamente. Como observamos neste capítulo, esse ganho de eficiência não causou perda de desempenho no modelo numérico, sendo que em alguns casos, foram registrados ganhos no tempo consumido.

4 *Caminhada Determinista do Turista em Meios Unidimensionais*

A ocorrência de um ponto crítico na caminhada do turista, delimita duas fases literalmente distintas, uma explorador local, na qual o caminhante, limita-se a visitar determinada região, e outra explorador global, que permite ao caminhante visitar todos os pontos do mapa. Esse fenômeno de transição, caracterizado por um aumento gradativo na memória μ, dado qualquer valor de N, recebe o nome memória crítica (μ_c). Primeiramente representada no trabalho de Rodrigo S. González [27], a transição de fase no meio unidimensional, foi realizada segundo uma transição encontrada no modelo de percolação, através de aglomerados condutores (elos), apresentada na referência [30]. Fundamentado no trabalho da referência [27], apresentamos alguns resultados de transição, usando o modelo determinista, com memória e com cpc, bem como, outra abordagem de transição, na qual a percolação é representada através do tamanho do período p dos ciclos de atratores. Verificamos também, que a memória ligada às imposições geométricas do meio, influenciam na existência ou não de alguns períodos de atratores. Através de experimentos numéricos, confirmamos que os ciclos de atratores, com períodos $2\mu+1$, $2\mu+2$ e $2\mu+3$, nunca ocorrem, e dessa forma, nós os classificaremos, simplesmente de períodos proibidos.

4.1 Percolação

Em sentido mais amplo, dizemos que a teoria da percolação é uma subdivisão da teoria de probabilidade, que lida com as propriedades de um meio aleatório. Seu conceito é útil para modelar a propagação de incêndios ou epidemias, fluxo de água em rochas porosas, fluxo de eletricidade em redes eléctricas, entre outros. Proposto por Broadbent e Hammersley em 1957 [30], a percolação é processo pelo qual um fluido se propaga em um meio aleatório. Nestes meios, a propagação ocorre de maneira não linear, onde uma transição brusca, conhecida como transição de percolação, separa dois regimes bem definidos (condutores e não condutores em meios desordenados).

A literatura dedicada à teoria de percolação não é exaustiva, apresentando uma diversidade de processos de percolação com aplicações práticas [31, 32, 33, 34, 35], e de acordo com [32], um processo de percolação também pode ser interpretado como um fluído em um meio aleatório.

4.2 Percolação em uma Rede Regular

Introduzimos o modelo de percolação de sítios (pontos) em uma rede regular, para expormos a ideia de transmissão ou de percolação, no modelo exploratório do turista. Seja uma rede quadrada, cujo sítios, condutores e não condutores, são distribuídos aleatoriamente sobre ela. Os sítios condutores ocorrem com uma probabilidade p e os não condutores com uma probabilidade $1-p$. O esquema da Fig. 4.1, exibe o modelo, dividindo um caixa quadrada em quadrados bem menores, para formar uma espécie de malha ou rede regular. O tamanho do sistema equivale a 16×16 células.

Nesse raciocínio, cada célula preta ocorre com probabilidade p, enquanto que as brancas ocorrem com probabilidade $1-p$. Quando $p = 0$, nenhum quadrado é pintado e a probabilidade de percolação é 0. Por outro lado, quando $p = 1$, todas as células são pintadas e o sistema percola sempre.

Seguindo a propriedade probabilística e acompanhando os dados da Fig. 4.1, vemos que para valores muito pequenos de p, poucos sítios são conectados, então há uma formação de aglomerados condutores, mas não o suficiente, para haver percolação do meio. Porém, parece evidente que a medida que p cresce, os sítios se aglomeram mais facilmente, aumentando assim a probabilidade do sistema percolar.

Definiu-se como probabilidade de percolação a probabilidade de um dado sítio, em uma rede muito grande ($N \gg 1$) pertencer a um aglomerado condutor. Desta forma, se $p = 1$, teremos um aglomerado que atravessa a rede de uma extremidade a outra (aglomerado ou *cluster* percolante). Portanto, se aumentamos p, a partir de $p = 0$ veremos que em um determinado valor $p = p_c$, ocorrerá a probabilidade crítica, abaixo da qual o sistema nunca percola e acima da qual o sistema percola sempre, gerando a transição de percolação. Este modelo é conhecido como percolação de sítios.

Podemos também considerar que cada célula contém um ponto que pode estar ligado a cada um de seus primeiros vizinhos com probabilidade p. Desta forma, alterando o valor de p alteramos a conectividade da rede. Em um valor $p = p_c$, ocorrerá a formação de uma aglomerado percolante, ligando as extremidades da rede. Este modelo é conhecido como percolação de elos.

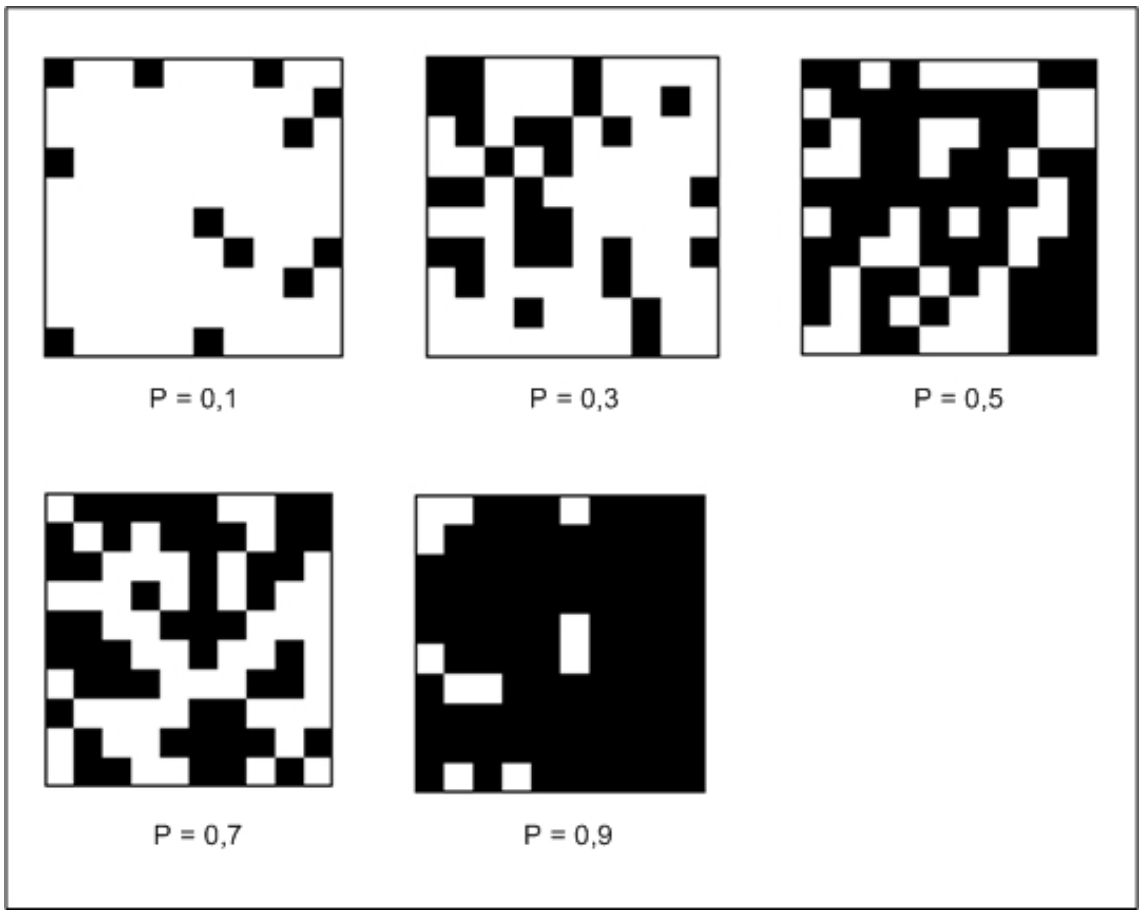


Figura 4.1: Exemplo de percolação por sítios, em uma rede regular 16×16, com $0,1 \leq p \leq 0,9$.

A percolação é uma transição de fase, uma vez que ela separa fases distintas em um determinado sistema. Caracteríza-se por apresentar mudanças bruscas de uma ou mais propriedades físicas, demonstradas por uma variação abrupta de um parâmetro de ordem. Um exemplo clássico de transição de fase, pode ser demonstrada comm as mudanças ocorridas nos estados físicos da água, que ocorrem mediante uma variação de temperatura ou de pressão [33], e neste caso, o parâmetro de ordem é a densidade do fluído.

4.3 Percolação na Caminhada do Turista

Na caminhada do turista unidimensional, o espaço pode ser compreendido como um segmento de reta, no qual a dinâmica do sistema se desenvolve. O movimento de um caminhante pode iniciar para a direita, ou para a esquerda (Fig. 4.2). A decisão da caminhada é traçada em direção à localização do ponto, que esteja à menor distância do ponto em que se encontra o caminhante num dado instante de tempo. Recordamos que o caminhante pode voltar à posições anteriormente visitadas, que não se encontram na memória de exclusão.

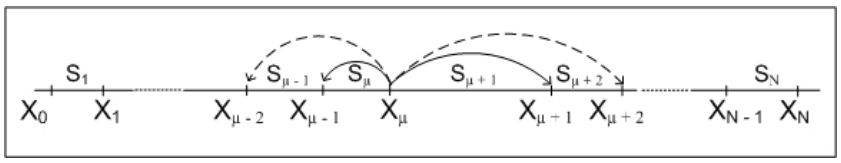


Figura 4.2: Num dado instante de tempo, o caminhante partindo do ponto X_μ, pode se deslocar para a sua esquerda, para o seu primeiro vizinho mais próximo $X_{\mu-1}$, e em seguida para o seu segundo vizinho mais próximo $X_{\mu-2}$, uma vez, estando na memória de exclusão, os pontos à sua esquerda, o caminhante pode se deslocar para a sua direita, para os dois primeiros vizinhos mais próximos, nesta sequência, $X_{\mu+1}$ e $X_{\mu+2}$.

Os resultados apresentados na referência [27], demonstrou que a topologia unidimensional, é oportuna para modelar a caminhada do turista, similarmente à um sistema de percolação de elos. Para isso, considerou-se as extremidades do segmento de reta, para delimitar os lados opostos do meio. Logo, estabeleceu-se que uma das extremidade, receberia o valor 1, primeira posição, enquanto a última posição receberia N. A idéia principal foi de lançar um caminhante do ponto inicial (1), e passando por todos os pontos, fazê-lo chegar a posição final (N), ocasionando assim, o transporte no meio.

4.3.1 A Existência do Ponto Crítico

Em se tratando de caminhadas do turista, determinadas situações, envolvendo a memória do sistema, necessitam ser elucidadas, por exemplo. Ao considerarmos o turista sem memória $\mu = 1$, teremos uma exploração localizada, na qual o caminhante se aprisionará em ciclos do tipo casal, isso impede uma transição imediata que só poderá ocorrer quando o par, atrator, se encontrar na extremidade final do meio.

Outro caso é se implementarmos o turista com memória $\mu = N$, caminhada auto-repulsiva, na qual o caminhante é obrigado a visitar todos os pontos através da imposição da regra do modelo. Neste caso, existe a formação de trajetórias com $t = N$, e não é possível finalizar a caminhada, levando o sistema a um estado final singular. Por último, no turista com memória $\mu = N - 1$, o caminhante, visitará todos os pontos, visto que, a dinâmica impõe a formação de atratores de períodos $p = N$.

Ainda sobre a referência [27], o autor não considerou a finalização das caminhadas em ciclos de atratores, sugerindo que a transição, na caminhada do turista, fosse realizada a partir da visitação a todos os pontos do mapa, independentemente dos pontos pertencerem a parte transiente ou a parte atrator, e usando o modelo determinista com condições abertas de contorno, confirmou que a probabilidade do caminhante sair da origem e chegar até N, é condicionada a um valor de memória μ, presente no intervalo ($1 < \mu_c \ll N - 1$), dado um determinado N, sendo

que $\mu_c \ll N$.

4.3.2 Resultados na Caminhada do Turista Unidimensional

A transição de fase, na caminhada do turista, ocorre tanto no meio com condição periódica de contorno, quanto no meio com as condições abertas de contorno [27], uma vez que estes, forneceram resultados equivalentes. Na Figura 4.3, apresentamos um esboço de caminhadas que percolaram em diferentes tamanho de sistemas, sobre uma configuração com $M = 1000$ mapas. A Tabela 4.1, fornece os valores de memória crítica para cada um dos N dado.

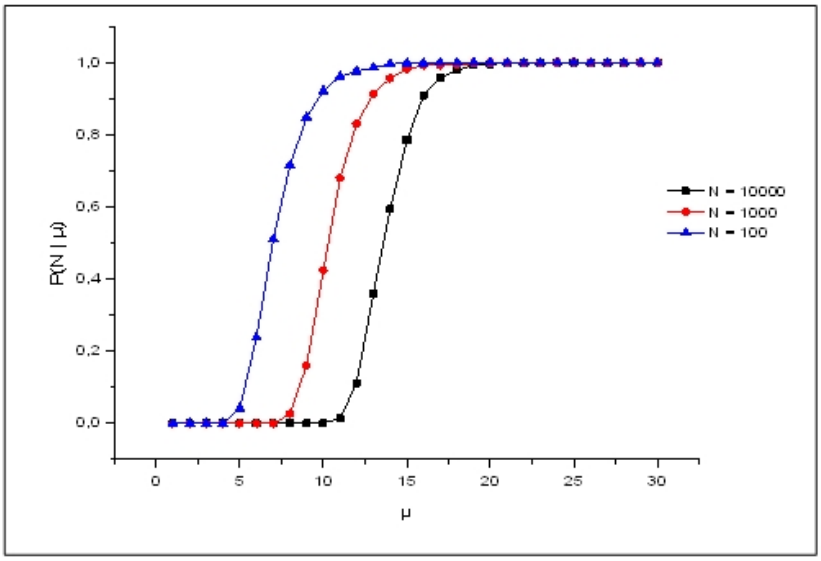


Figura 4.3: Médias de caminhadas que percolaram em diferentes sistemas, no modelo com condição periódica de contorno.

Notamos que para cada valor de N, há um valor crítico no qual a probabilidade de percolação aumenta em mais de 50%. Após μ_c, há uma tendência para que a probabilidade de visitar todos os N pontos seja $P(N|\mu) = 1$.

Considerando outro modelo do turista com memória $\mu \geq 2$, sobre $M = 1000$ realizações, verificamos que a partir da memória crítica (ver Tabela 4.1), as caminhadas se estabilizam em ciclos de atratores de período $p = N$. A Figura 4.4, mostra uma transição semelhante a anterior, porém, neste caso, o período p é o causador da mudança de fase do sistema.

A transição de fase, no modelo exploratório do turista, separa dois regimes bem diferentes. Em um deles (Fig. 4.3), o caminhante, mesmo com a imposição determinista, consegue visitar todos os lugares do mapa unidimensional, utilizano um valor muito baixo de memória. Em outro caso (Fig. 4.4), mesmo se aprisionando em ciclos periódicos, o caminhante explora todos

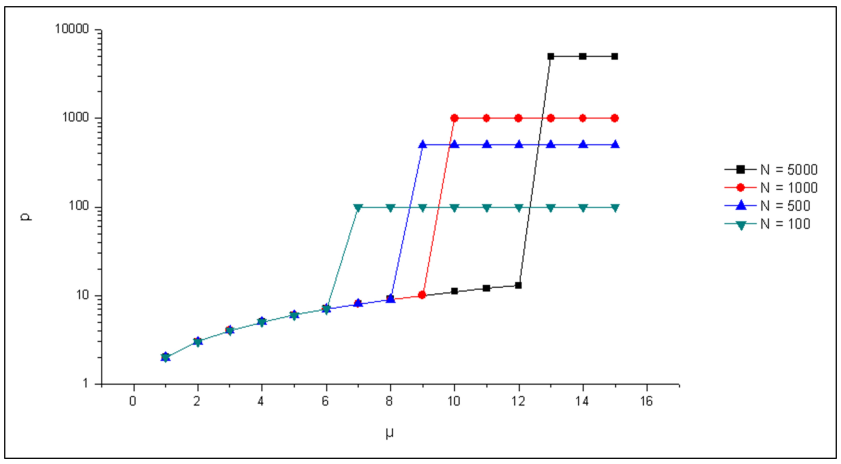


Figura 4.4: Formação de ciclos de atratores de períodos p_{max} para $\mu \geq \mu_c$.

N	μ_c
100	6.6
500	9.0
1000	10.0
5000	12.2
10000	13.3

Tabela 4.1: Valores de μ_c obtidos para $N = 100$, 500, 1000, 5000 e 10000.

os pontos do sistema, ao atingir uma memória no intervalo $(1 < \mu_c \ll N-1)$, apesar da memória ser $\mu_c \ll N$.

4.4 Novas Descobertas em Meios Unidimensionais

Quando se trata de sistemas dinâmicos, meios unidimensionais favorecem a investigação de qualquer fenômeno, pois são modelos de mais fácil observação. As propriedades geométricas do meio e a regra determinista do modelo, influenciam na formação de atratores de forma que os ciclos de atratores de períodos $p = 2\mu+1, 2\mu+2, 2\mu+3$ nunca são gerados, nos confirmando que existe uma combinação de passos impedida de serem executadas, a este fato denominamos de períodos proibidos.

4.4.1 A Frequência de Atratores de Períodos Mínimo e Máximo

Na caminhada do turista, atratores surgem devido à dinâmica determinista do sistema, unidos à distribuição dos pontos e ao tamanho da memória (μ), empregados. É a memória quem determina o tamanho do menor período, formado em uma trajetória, porém o maior período não podemos estimar. Dessa forma, os atratores mais comuns, para qualquer μ, possuem período

$p = \mu + 1$, também denominado de período mínimo p_{min}, e o número de pontos que formam o menor período é também mínimo $n_s = \mu + 1$, como ilustrados na Fig. 4.5. Por analogia, classificaremos de p_{max} os atratores de período $p = N$, igualmente com $n_s = N$.

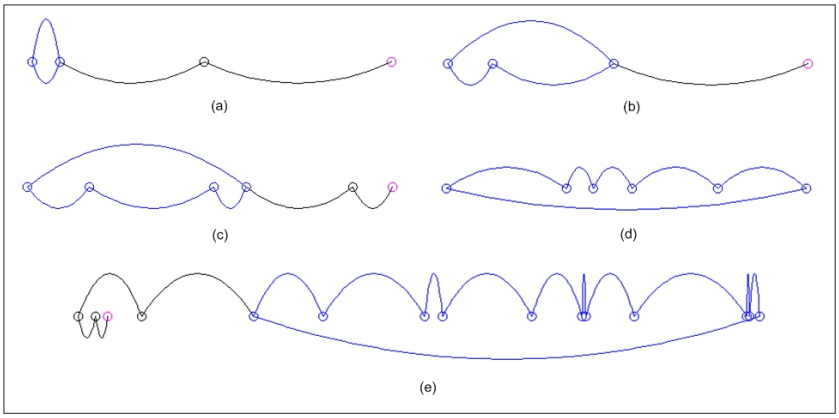


Figura 4.5: Exemplos de diferentes trajetórias com a formações de transientes e atratores de períodos $p = \mu + 1$, para determinados valores de μ. (a): período 2 com $\mu = 1$, (b): período 3 com $\mu = 2$, (c): período 4 com $\mu = 3$, (d): período 6 com $\mu = 5$, (e): período 11 com $\mu = 10$.

Dos experimentos numéricos analisados em várias dimensionalidades, verificamos que os períodos p_{min} ocorrem com maiores frequencias do que os demais tipos de períodos. Isso ocorre devido a pouca combinação de movimentos que um caminhante precisa executar, para formar um período mínimo, e que veremos detalhadamente na seção 4.4.3.1. Por se tratar de um meio cuja dinâmica permite percolação, é esperado um aumento de trajetórias que se estabilizem em atratores de período p_{max}. Sendo assim, a medida que μ se aproxima do ponto crítico, a frequência de atratores p_{min} tendem a diminuir, enquanto que a de atratores p_{max} aumentam, esse fato só se aplica ao turista unidimensional, devido ao aperecimento da transição. No gráfico da Fig. 4.6, com $M = 1000$ realizações numéricas, mostramos a tendência das trajetórias atigem períodos p_{max}, a medida que μ se iguala ao μ_c, lembramos aqui que para $N = 100$ o ponto crítico é aproximadamente ($\mu_c \approx 7$).

4.4.2 Atratores Proibidos

Um atrator é o ponto ou região, no qual um sistema dinâmico eventualmente se estabiliza. Como vimos em capítulos anteriores, a dinâmica do turista, sempre leva a trajetórias compostas inicialmente de um tempo de transiente, de tamanho t, e de um atrator periódico de período (p), no qual o caminhante é aprisionado. Os atratores surgem devido à dinâmica da caminhada, combinados com as limitações geométricas do meio e com a memória μ introduzida. Dessa forma, condições iniciais diferentes, geram comportamentos diferentes e determinam a existência ou não de alguns períodos de atratores.

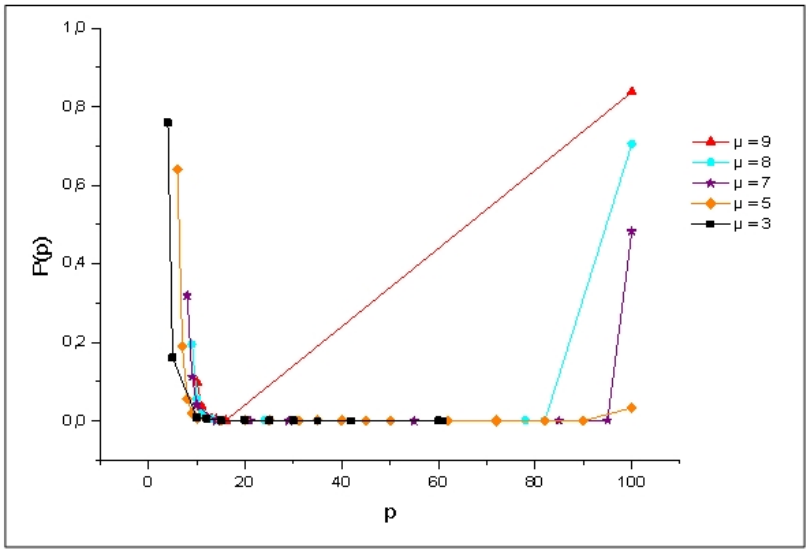


Figura 4.6: Experimentos numéricos realizados com $N = 100$, esquematizando que à medida que μ se aproxima do valor crítico, a ocorrência de atratores com período $p = \mu + 1$ diminui, aumentando a ocorrência de períodos $p = N$.

Nos trabalhos apresentados por Gilson F. Lima et. al. [17, 23], já verificava-se a proibição de atratores ímpares maiores que $p = 3$ e iguais a $p = 6$, quando este realizou experimentos numéricos do turista determinísta, com memória $\mu = 2$. Segundo Lima [23], a não formação de atratores ímpares maiores que 3, correspondia às restrições geométricas do meio, e que um ciclo 6 nunca poderia ser formando, pelo fato de não ser possível unir dois atratores $p_{min} = 3$ no modelo determinista, com memória $\mu = 2$.

Ainda, concernente à referência [23], o autor observou que os resultados das caminhadas, cujas trajetórias não formavam atratores ímpares maiores que $p = 3$ e iguais a $p = 6$, se restringiram, exclusivamente, para o caso $\mu = 2$, sendo, portanto, a nossa descoberta inédita, quando verificamos que esse comportamento, no meio unidimensional, é pertinente a qualquer valor de memória, e que os ciclos de atratores com períodos $(2\mu+1, 2\mu+2, 2\mu+3)$, nunca se formam.

4.4.3 Resultados

Na condição em que o sistema determinista do turista obedecia a uma certa regularidade, que ao longo das caminhadas, as trajetórias formadas a partir de atratores de período $p = 2\mu$, cujo ciclos que se seguem $2\mu+1, 2\mu+2, 2\mu+3$, não se realizavam, chamou a nossa atenção. Essa eventualidade motivou-nos a averiguar as combinações possíveis e não possíveis de passos, que um caminhante precisa executar ao explorar o meio unidimensional, que preveem a

realização, ou não de determinados períodos de atratores.

Diversos experimentos numéricos, com o modelo do turista unidimensional, foram realizados para diferentes valores de μ. A melhor maneira de observamos o comportamento dessas trajetórias foi submeter os dados de saída do modelo, que compreendem as coordenadas dos pontos, obtidos na formação de transientes e atratores, a um programa capaz de visualizar tais informações[1].

No programa de visualização dos atratores, são representados apenas os pontos inerentes às trajetórias (t e p), sendo dispensada a imagem dos demais pontos, componentes do meio. Para explicarmos, com maiores detalhes, as características de tais periodicidades, dividiu-se esta seção em duas subseções, nas quais faremos alusão aos atratores formados primeiramente com $p = \mu + 1$ e com $p = 2\mu$, dos quais derivam-se as demais ocorrências de atratores. Ambos os tipos de períodos, são consequências diretas da memória μ e da decisão de deslocamento do caminhante, neste caso, para a direita ou para a esquerda.

4.4.3.1 A Formação de Atratores do tipo $p = \mu + 1$

A formação de um período $p = \mu + 1$, ou período mínimo, é prevista quando um "salto"é realizado, a partir do instante que o caminhante percorreu μ passos, ou seja, estando o caminhante no último ponto da região atratora, este retorna ao primeiro ponto do atrator, que se encontra a uma distância $\mu - 1$ passos do mesmo.

Um salto pode ser definido, como um deslocamento, necessário, que um caminhante deve executar partindo do ponto i para o ponto $i \pm n$, para formar um ciclo ($p = \mu + 1$), onde $n \geq 2$. Assim, ao realizar um salto, o turista não se desloca para o seu vizinho, imediatamente a frente, ele simplesmente pula sobre os $\mu - 1$ pontos da janela de memória, retornando ao ponto de partida, fechando um ciclo. Este procedimento pode ser visualizado na Fig. 4.7.

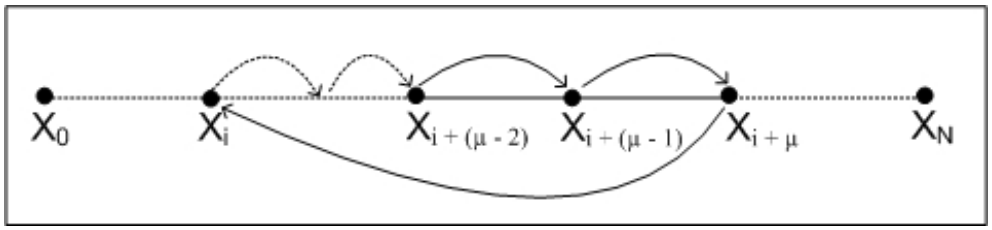


Figura 4.7: Passos que um caminhante executa para formar um atrator com período $p = \mu + 1$.

[1] O programa visualizador, utilizado neste trabalho de pesquisa, é definido no Apêndice 1.

4.4.3.2 A Formação de Atratores do tipo $p = 2\mu$

Para uma trajetória gerar um ciclo $p = 2\mu$, cuja geometria compara-se a de um polígono simples, onde ainda não há cruzamentos de pontos no atrator, o caminhante precisa se deslocar do ponto i, para o ponto $i \pm 2$, já excluído da janela de memória μ, ocorrendo assim o primeiro salto. O segundo deslocamento (salto), é realizado ao fechar o ciclo, conforme podemos verificar no esquema da Fig. 4.8.

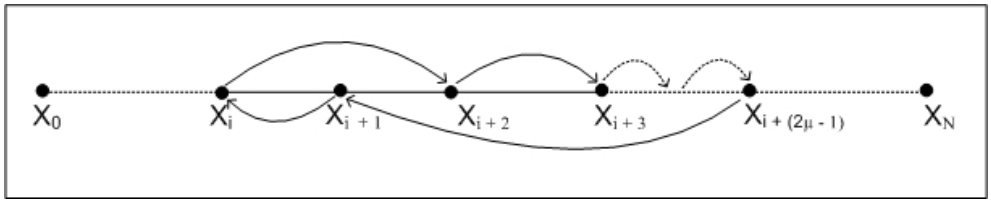


Figura 4.8: Passos que um caminhante executa para formar um atrator com período $p = 2\mu$.

Ressaltamos que esta é a única maneira possível de se construir o ciclo $p = 2\mu$, em uma dimensão. O primeiro salto só é realizado, sobre apenas um ponto no atrator, executando também o primeiro deslocamento de posição, sempre oposto ao sentido que se encontra o caminhante.

Ainda podemos afirmar que, em decorrência da regra determinista, o caminhante pode saltar no mínimo $\mu - 1$ pontos do atrator, e que este sempre fecha no primeiro ponto, onde se iniciou a caminhada.

4.4.3.3 A Não Formação de Atratores com Período $2\mu+1, 2\mu+2, 2\mu+3$

No sistema determinista do turista, a dinâmica de visitar um ponto mais próximo, que ainda não tenha sido visitado nos últimos μ passos anteriores, demarca que, ao longo do tempo, o caminhante ocupe sequencialmente pontos do espaço, gerando os transientes e os ciclos de atratores. Ao analisarmos as trajetórias no modelo unidimensional, verificamos que os atratores de períodos $2\mu+1, 2\mu+2, 2\mu+3$, não são possíveis de serem construídos. A ideia então é entendermos o processo de formação dos atratores, fora desse intervalo, para podemos explicar porque que a reunião de pontos no mapa, e a regra determinista, não gera determinados tipos de atratores.

Ao analisarmos o atrator de período $2\mu+4$, o primeiro ciclo formado a partir do intervalo proibido, verificamos que este é um atrator do tipo complexo ($p > n_s$), constituído pela união de dois atratores do tipo simples ($p = n_s$), localizados em cada extremidade do atrator. Todo o atrator simples, especificado para este caso, deve pertencer ao intervalo $\mu+1 \leq p \leq 2\mu$. Tal fato pode ser observado na Fig. 4.9, onde obtivemos alguns modelos de atratores de período

$p = 2\mu + 4$, a partir de diferentes grandezas de memória.

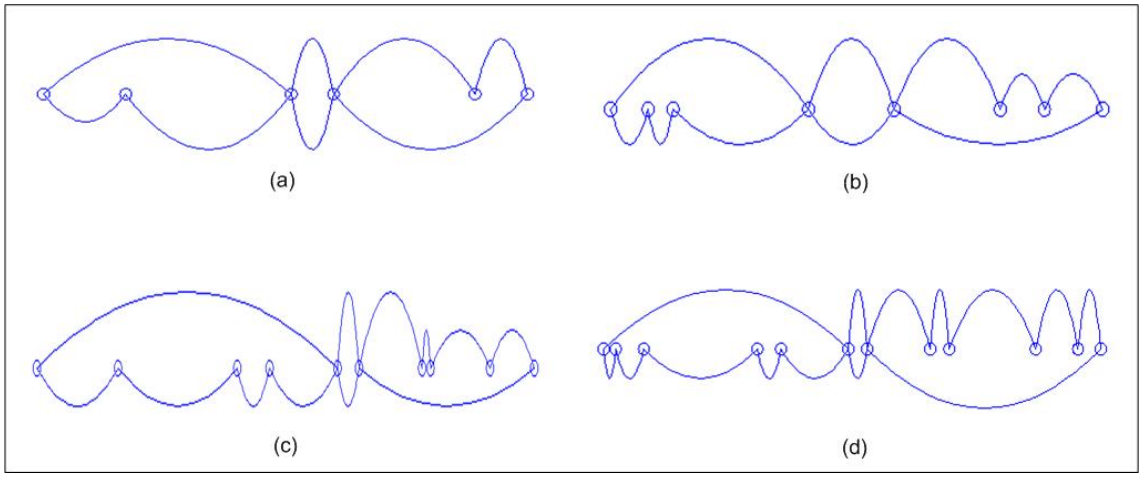


Figura 4.9: Formação de atratores cujo $p = 2\mu + 4$: (a): $p = 8$ para $\mu = 2$, (b): $p = 10$ para $\mu = 3$, (c): $p = 12$ para $\mu = 4$ e (d): $p = 14$ para $\mu = 5$.

Examinando o esquema da Fig. 4.9, notamos que a ocorrência de um atrator $p > n_s$, está associada a dois fatores intrínsecos: a colocação (ordem) dos pontos no meio unidimensional e ao ponto de partida (inicial) da caminhada. Na Fig. 4.10, abaixo, mostramos em que ordem os pontos devem ser percorridos, para que se produzam ciclos de atratores com período $p = 2\mu + 4$, para o modelo do turista com $\mu = 2$, da Fig. 4.9(a), por exemplo.

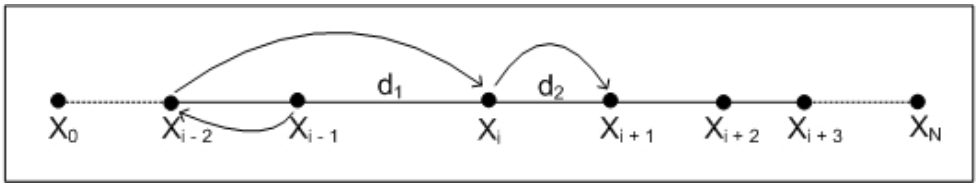


Figura 4.10: Passos de uma trajetória para formar um atrator do tipo $p = 2\mu + 4$, para $\mu = 2$.

Observamos que a formação do atrator (a), inicia-se no ponto x_{i-1}, com um deslocamento à esquerda para o ponto x_{i-2}. Posteriormente, o caminhante realiza um salto sobre o ponto x_{i-1}, que ainda se encontra na janela de memória, mundando a direção do curso, e indo em direção ao ponto x_i. Havendo outras condições de distância, no modelo com memória $\mu = 2$, é comum que o caminhante, estando no ponto x_i, retorne para o ponto x_{i-1}, que já não estará mais na memória, e finalize a caminhada, fechando um ciclo de atrator onde $p = \mu + 1$. No entanto, neste caso específico, a distância d_2, entre o ponto atual e aquele imediatamente a direita, x_{i+1}, deve ser menor do que a distância d_1, para que o caminhante vá até o ponto x_{i+1} e prossiga até o ponto x_{i+3}, levando à formação do primeiro laço (x_{i+1}, x_{i+2} e x_{i+3}), de uma das extremidades do atrator. Todavia, o atrator somente estará concluído, quando o caminhante retornar para o ponto x_{i-1}, o que acontece, quando este atinge ao ponto x_i.

Denota-se então, que estas características de deslocamento formam os atratores de períodos $2\mu+4$, onde dois atratores de períodos $\mu+1$, são unidos através de 2 passos (um passo para ir, outro passo para voltar), produzindo o cruzamento dos pontos, na junção de ambos atratores. Dessa forma, podemos deduzir que $1(\mu+1)+1(\mu+1)+2=2\mu+4$, e paralelamente concluirmos que o menor ciclo de atrator permitido possuirá no mínimo um período $p=2\mu+4$. Assim, as regiões proibidas, representadas pelos períodos $2\mu+1, 2\mu+2$ e $2\mu+3$, são combinações de passos, que não podem ser percorridos.

4.4.3.4 A Não Formação de Atratores Impares Maiores que 3 com $\mu=2$

Como verificamos nos experimentos numéricos realizados, na caminhada unidimensional, toda a formação de ciclos de atratores de períodos superiores a $2\mu+4$, de ordem par, são constituídos de dois ciclos $\mu+1$ (nas extremidades do atrator) conectados por n pontos intermediários. Já, para a formação de ciclos de atratores de ordem impar (Fig. 4.11), a localização de dois pontos, mutuamente próximos, na bacia do atrator, determina que mais um passo, extra, se faça necessário para o fechamento de um dos ciclos do atrator, essa volta extra, que o caminhante realiza, duas vezes, sobre o mesmo par de pontos, é o que especifica um ciclo impar Fig. 4.12. Essa regra se aplica para qualquer tamanho do sistema, independentemente do valor de memória μ utilizado.

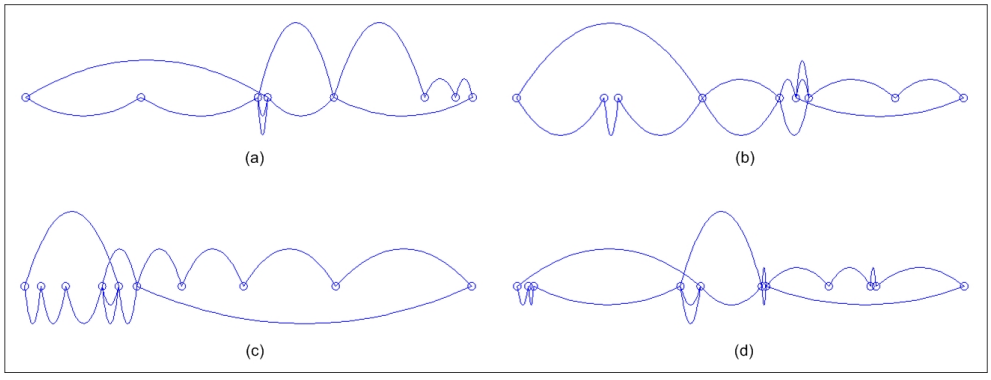


Figura 4.11: Passos de uma trajetória para formação de atratores de período ímpares: (a): $p=11$ para $\mu=3$, (b): $p=13$ para $\mu=3$, (c): $p=13$ para $\mu=4$ e (d): $p=15$ para $\mu=4$.

Para o caso do turista com memória $\mu=2$, o fato da memória ser muito pequena, implica na impossibilidade da execução da volta extra (x_{i+1}, x_i e x_{i+2}), como mostrado na Fig. 4.12, ou seja, quando o caminhante partindo do ponto x_{i+2}, atinge o ponto x_i, a distância d_1 sendo menor do que a distância d_2, obriga o caminhante retornar para o ponto x_{i+2} que já não se encontra na janela de memória de exclusão, fechando assim, um ciclo de período $p=\mu+1$.

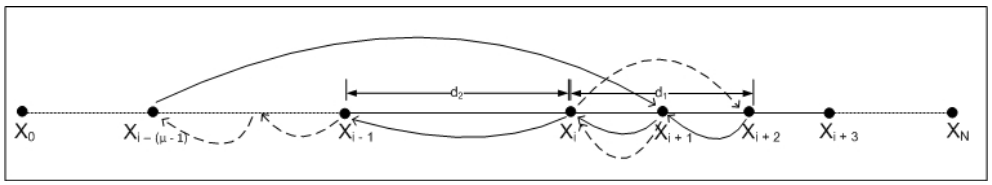


Figura 4.12: Passos de uma trajetória para formação de atratores de período ímpares.

4.5 Conclusão

Neste capítulo mostramos que existe uma descontinuidade caracterizada por uma variação abrupta na curva que separa os dois regimes (localizados e estendidos) apresentados na caminhada do turista, para um dado valor de memória $\mu_c = \log_2 N$. No entanto, é necessário que a dinâmica determinista permita que a memória μ se aproxime de μ_c, para haver exploração total no meio.

Verificamos que no modelo determinista, unidimensional, com condição periódica de contorno e sem esta, os resultados referentes a transição de fase são equivalentes. E através da análise dos tamanhos dos períodos, verificamos outra metodologia de transição, no modelo do turista, apresentada pelos atratores de períodos da ordem de N.

Ainda neste Capítulo, mostramos que existem mais ciclos proibidos em um meio unidimensional, do que já foi apresentado até o momento. A geometria espacial e a memória μ empregados, são fatores determinantes para a formação e não formação de alguns períodos de atratores, o que sugere que condições iniciais diferentes, geram comportamentos diferentes e implicam na existência ou não de alguns períodos de atratores.

Os atratores periódicos, encontrados na caminhada determinista do turista, obedecem padrões cíclicos de movimentação. Quando se tenta representar a forma desse conjunto de comportamentos característicos, descobre-se que essas se assemelham à geometria dos polígonos. Dessa maneira, classificamos os atratores do modelo, segundo a fronteira que delimita tal figura geométrica, e chamamos de atratores simples, aquele cujo o período p é igual ao número de pontos que o compõem ($p = n_s$) e de atratores complexos, quando dois ciclos simples são ligados por determinados pontos intermediários (n), formando os períodos que se cruzam, e neste caso ($p > n_s$).

Então, através de experimentos numéricos, reparamos outro padrão de comportamento, quanto aos atratores de período p, na caminhada do turista unidimensional. Para um dado valor de memória μ, os ciclos formandos por períodos do tipo $\mu+1$ a 2μ, se assemelham a polígonos simples, depois os ciclos $2\mu+1, 2\mu+2, 2\mu+3$, são proibidos, e a partir daí, qualquer ciclo for-

mado passa a ser do tipo complexo. Contudo, ao atingir um período $p = N$ temos novamente um ciclo do tipo simples, e ciclos posteriores serão sempre complexos, formados por períodos do tipo ($p > N$).

E ainda para o caso das formações de ciclos de atratores, conseguimos demonstrar o porquê da não formação de alguns períodos de atratores. Verificamos que a distribuição dos pontos no meio unidimensional, especificamente, o ajuntamento de dois pontos muito próximos, leva à formação de atratores ímpares, enquanto que apenas para o modelo do turista com memória $\mu = 2$, a formação de atratores ímpares não se realiza devido ao significativo valor da memória. Contudo, podemos afirmar que no meio unidimensional, o modelo com memória $\mu = 2$ apresenta um comportamento diferentes, do restante, no que abrange a formação de atratores.

5 *Aplicação no Hiperespaço*

Os conceitos e métodos tratados no modelo unidimensional serviram de inspiração para outras ocorrências, como é o caso dos sistemas d dimensionais do turista, para os quais consideramos as mesmas grandezas introduzidas no capítulo anterior. Dessa forma, este capítulo propõe-se a explorar os resultados numéricos referentes à caminhada do turista, na visão determinista, para os quais definiu-se um conjunto com muito mais pontos do que se tinha apresentado até então.

Ocorre que os primeiros estudos sobre a caminhada determinista do turista também enfrentaram restrições de recursos computacionais, devido a época em que foram abrangidas. Todavia, mesmo com a evolução de tais recursos, como o surgimento dos processadores de vários núcleos (*multicore*) e a ultrapassagem da barreira dos 32 bits, o que possibilita equipar computadores com quantidades de memória superiores a 4GB, o *software* de simulação da caminhada do turista ainda sofre com a escassez de recursos. A quantidade de memória DRAM necessária para armazenar os resultados das caminhadas é grande e ainda não se tornou popular. Além disso, as estruturas de dados do tipo matriz são ineficientes para tratar o problema, devido à formação de matrizes esparsas. No entanto, com o desenvolvimento da lista hierárquica-HL, tornou-se possível superar tais restrições, como demonstramos no Capítulo 3.

Por essa razão, o motivo específico para examinarmos o problema da caminhada determinista, foi explorar o modelo com memória μ da ordem de grandeza de N, no espaço d dimensional. E, ao final, cotejar com os resultados anteriormente alcançados, demonstrados na caminhada do turista, com o μ pequeno.

5.1 Caminhada do Turista no Espaço d Dimensional

No geral, as análises da dinâmica do turista, no espaço não unidimensional, são mais críticas, uma vez que existem diferenças qualitativas entre os casos $d = 1$ e $d = 2$. Porém, para dimensões superiores, o comportamento é similar ao obtido em $d = 2$. Numa primeira ob-

servação, verifica-se que algumas formações de ciclos de atratores são provenientes de uma mistura ou de uma justaposição de ciclos primários, oriundos do aumento do valor da memória μ.

5.2 Resultados para o Modelo Bidimensional

O nosso objetivo foi explorar o modelo determinista bidimensional (com condições periódicas de contorno), com a memória μ da ordem de grandeza de N, verificando o comportamento do sistema. O primeiro passo foi examinar a exploração do turista na paisagem, na tentativa de encontrar no modelo uma transição entre regimes, onde as caminhadas, ao atingirem um valor crítico de memória, permitissem percorrer todos os pontos do sistema, semelhante ao que foi observado no modelo unidimensional (Capítulo 4). Para essa realização foram necessárias várias simulações numéricas, com as quais exigiram-se elevadas taxas de armazenamento de dados, devido a um rápido crescimento das informações geradas, sendo esta mais uma razão da mudança na estrutura de dados do modelo (Capitulo 3), para que fosse possível tratar tais informações.

5.2.1 Distribuição de Tempos de Transiente

A dinâmica da caminhada determinista do turista evolui a intervalos discretos de tempo, definindo o comprimento do transiente t, ou seja, quantos passos o caminhante andou até entrar em um ciclo de atrator. A distribuição de probabilidade marginal de t, no intervalo de memória considerado $25 \leq \mu \leq 900$, com $N = 1000$, pode ser observada na Fig. 5.1.

Ao observarmos os resultados, percebemos que enquanto $\mu \leq 50$, o caminhante não consegue gerar transientes do tipo $t = O(N)$. Todavia, ao utilizarmos $\mu = 100$, o caminhante não apenas consegue gerar transientes deste tipo, como também gera transientes maiores que N. Porém, podemos observar que a ocorrência de transientes pequenos são mais frequentes independente do valor atribuído a μ.

É interessante observar que mesmo aumentando o valor da memória, esse comportamento se repete para todos os valores de μ analisados. Isso pode ser explicado pela transposição entre um regime de exploração local, obtido com pequenos valores de memória, para um regime de exploração global. No primeiro, a caminhada tende a entrar em um atrator pela ausência de memória suficiente, provocando uma exploração incompleta (parcial) do meio, caracterizando caminhadas com tamanho de transientes e de períodos pequenos. Por outro lado, no regime de

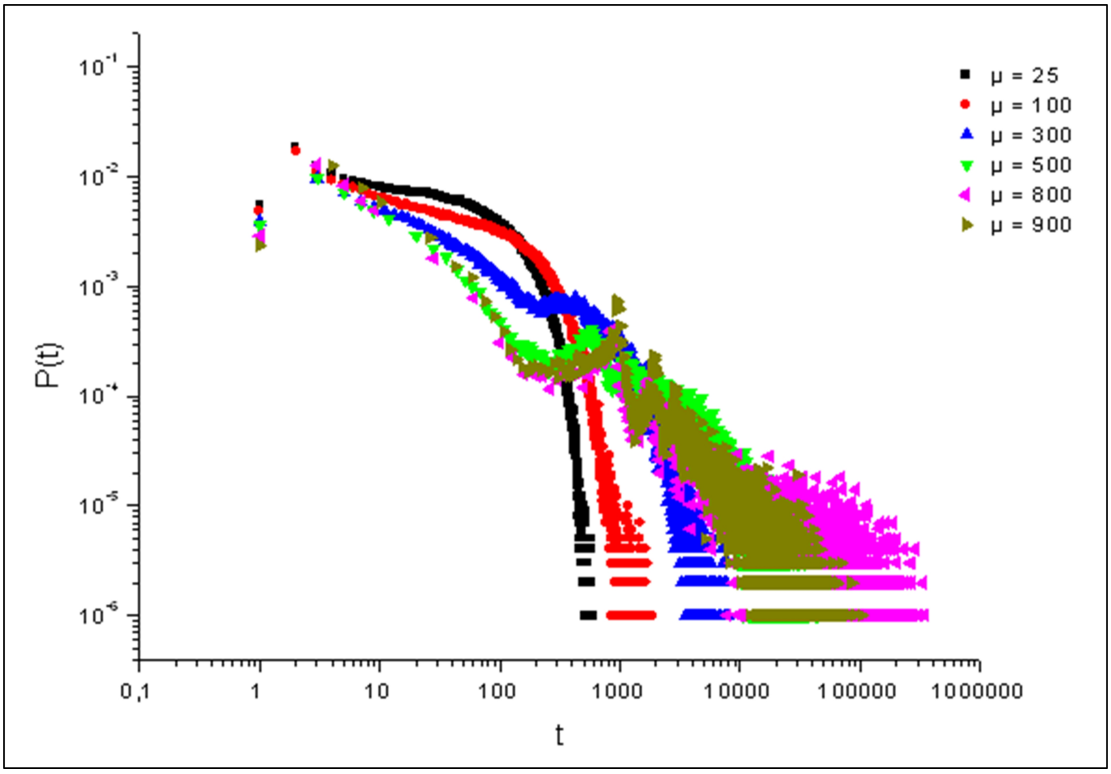


Figura 5.1: Distribuição de probabilidade de tempos de transiente para $N = 1000$ e $25 \leq \mu \leq 900$, obtidos a partir de simulações com $M = 1000$ realizações.

exploração global, o caminhante, por possuir um valor elevado de memória, tende a gerar períodos com valores de n_s elevados, reduzindo a quantidade de pontos disponíveis para a formação do transiente.

É importante observar que, a partir de $\mu = 300$, há uma mudança de comportamento das curvas obtidas, caracterizada por uma oscilação de frequência à medida que o valor de μ aumenta. Esse comportamento se deve à estabilização, temporária, da quantidade de sítios explorados com o aumento de μ. Tal característica pode ser observada na Fig. 5.2, que apresenta a média de sítios visitados ($\langle t + n_s \rangle$) para diferentes valores de N e μ.

Ao analisarmos o gráfico da Fig. 5.2, percebemos que em mapas cuja quantidade de sítios é pequena, $N \leq 100$, à medida em que o valor de μ aumenta, também aumenta a média de sítios visitados. Todavia, à medida em que aumentamos a quantidade de sítios por mapa, percebemos que o espaço explorado pelo caminhante permanece estável em um determinado intervalo de μ. A largura deste intervalo aumenta à medida em que também ampliamos a quantidade de sítios.

Esse comportamento demonstra que, à medida em que aumentamos a quantidade de sítios, a caminhada do turista apresenta três fases de exploração, independente do valor de N adotado, caracterizadas por dois regimes distintos. Na primeira fase, a quantidade de sítios visitados é muito baixa. Porém, à medida em que há um aumento unitário da memória, o caminhante

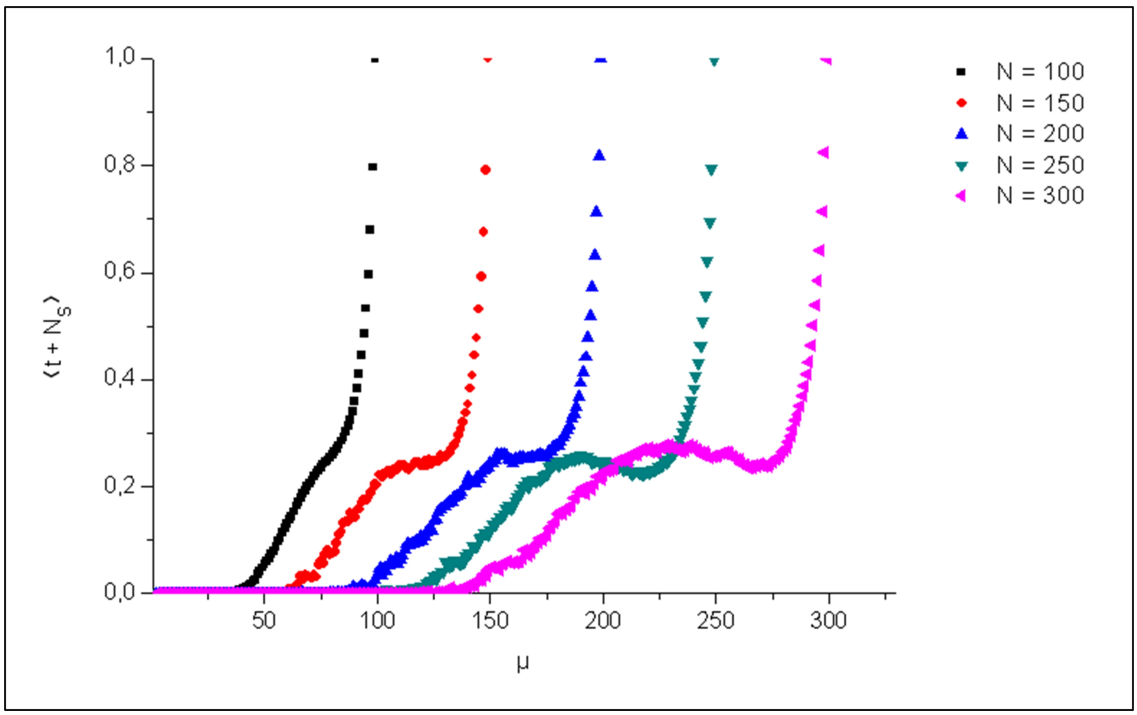


Figura 5.2: Média de sítios visitados $\langle t + n_s \rangle$ em função de μ para sistemas com $100 \leq N \leq 300$.

aumenta o espaço explorado obedecendo uma função exponencial. Já na segunda, mesmo com a ampliação de μ, não há ampliação significativa na quantidade de sítios explorados, fato esse que deixa de existir na terceira fase em que a quantidade de sítios explorados pelo caminhante aumenta, obedecendo novamente uma função exponencial, levando à exploração total do meio quando $\mu \approx N$.

Ao analisarmos os gráficos da Fig. 5.1 e da Fig. 5.2, percebemos que, na segunda fase de exploração do meio, a variação nos tempos de transientes gerados é máxima, sendo assim, podemos concluir que, para $\mu = 700$, ocorre o maior tempo de estabilização do meio, onde, mesmo com o aumento da memória, o caminhante não amplia a quantidade de sítios visitados.

A Fig. 5.3 expõe a distribuição de probabilidades de transiente em sistema em que $N = 100$. Percebemos que a formação de transientes não depende diretamente da grandeza N, e sim, da relação $N \times \mu$.

5.2.2 Distribuição de Período de Atratores

A distribuição de período de atratores, para $N = 1000$, pode ser visualizada na Fig. 5.4, onde $25 \leq \mu \leq 900$. Analisando a Fig. 5.4, percebemos que há três comportamentos distintos na frequência de formação de período de atratores. Para o primeiro grupo, em que $\mu < 800$, a formação de atratores de período $\mu + 1$ são mais comumente encontrados, visto que o valor

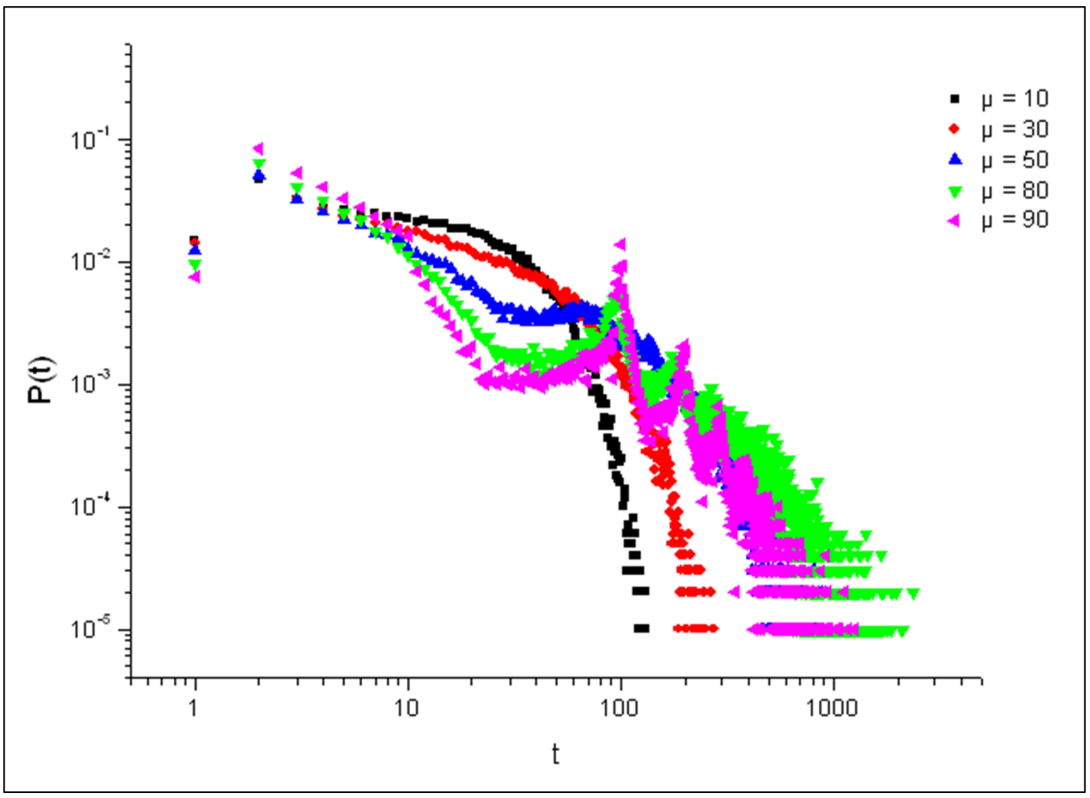


Figura 5.3: Distribuição de probabilidade de tempos de transiente para $N = 100$ e $25 \leq \mu \leq 900$, obtidos a partir de simulações com $M = 1000$ realizações.

da memória utilizada não é suficiente para aprisionar o caminhante em atratores de períodos de $O(N)$. Todavia, ao considerarmos $\mu = 800$, obtemos um cenário em que tanto a formação de atratores do tipo $p = \mu + 1$, quanto aqueles em que $\mu + 1 \leq p \leq N$, são frequentes. Posteriormente, onde $\mu > 800$, obtemos um terceiro cenário em que os atratores do tipo $p = N$ são mais comuns.

O comportamento descrito anteriormente pode ser melhor observado na Fig. 5.5, onde apresentamos isoladamente os resultados obtidos com $\mu \in \{600, 800, 900\}$ cujos eixos encontram-se na escala linear, o que permite analisar os três comportamentos distintos na formação atratores.

Através da Fig. 5.5, percebemos que a partir de $\mu = 800$, onde se encontra a zona transitória, de predominância de atratores do tipo $p = \mu + 1$, para a outra do tipo $p = N$, todos os atratores formados são do tipo $\mu + 1 \leq p \leq N$. Sendo assim, somente com valores de memória inferior ao da zona transitória, é que conseguimos formar atratores com $p > N$. Esse fato pode ser comprovado na Fig. 5.6, que demonstra os maiores períodos de atratores em diferentes valores de N.

Ao analisarmos o gráfico da Fig. 5.6.c, percebemos que ao utilizarmos $\mu = 600$, há a formação de atratores em que $p \approx 36.000$, caracterizando a existência de atratores do tipo $p \geq N$.

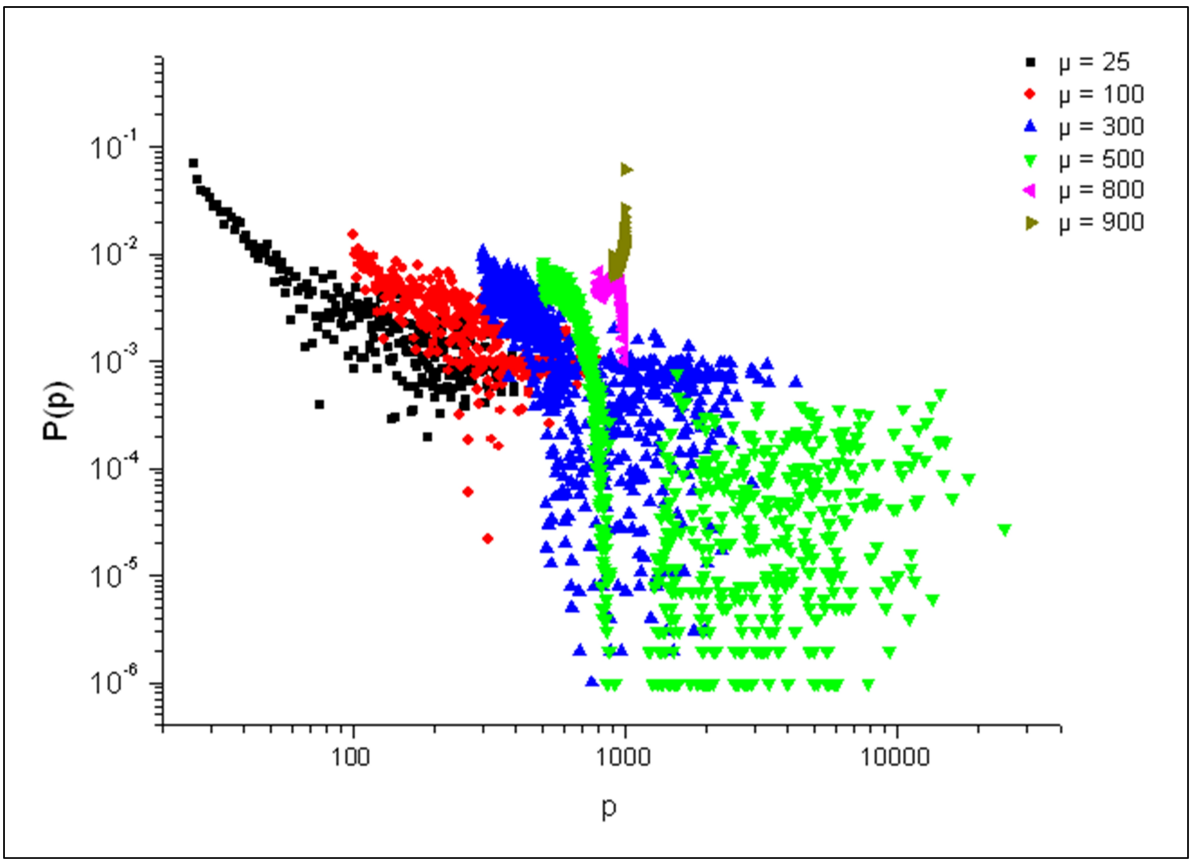


Figura 5.4: Distribuição de período de atratores para $N = 1000$ e $25 \leq \mu \leq 900$, obtidos a partir de simulações com $M = 1000$ realizações.

Todavia, à medida que ampliamos o valor da memória utilizada, passamos a um cenário em que os atratores formados são do tipo $p \leq N$. Essa mesma configuração pode ser observada para $N = 100$ e $N = 500$, nas Figs. 5.6.a e 5.6.b, respectivamente. Em ambas as configurações, esse comportamento também ocorre, indicando que tal característica não depende apenas do valor de N, mas sim da relação entre o número de sítios do meio desordenado e o valor adotado para a memória do caminhante.

Essa característica assemelha-se à observada no espaço unidimensional. Todavia, quando $d = 1$, há uma definição clara de um valor de memória μ_c para o qual é possível visitar todos os pontos do sistema. Assim, o atrator de maior período é gerado ao utilizarmos um valor de memória ligeiramente inferior àquele estabelecido como μ_c. No entanto, no espaço bidimensional, não há um valor de μ que possa ser considerado crítico pois, como observamos na Fig. 5.2, há diferentes cenários de exploração do meio, ao utilizarmos diferentes valores de N e μ.

Vale observar que, ao obtermos atratores em que $p \geq N$, não é necessário que o caminhante tenha percorrido todo os sítios do mapa. Podemos verificar esta afirmação pelo gráfico da

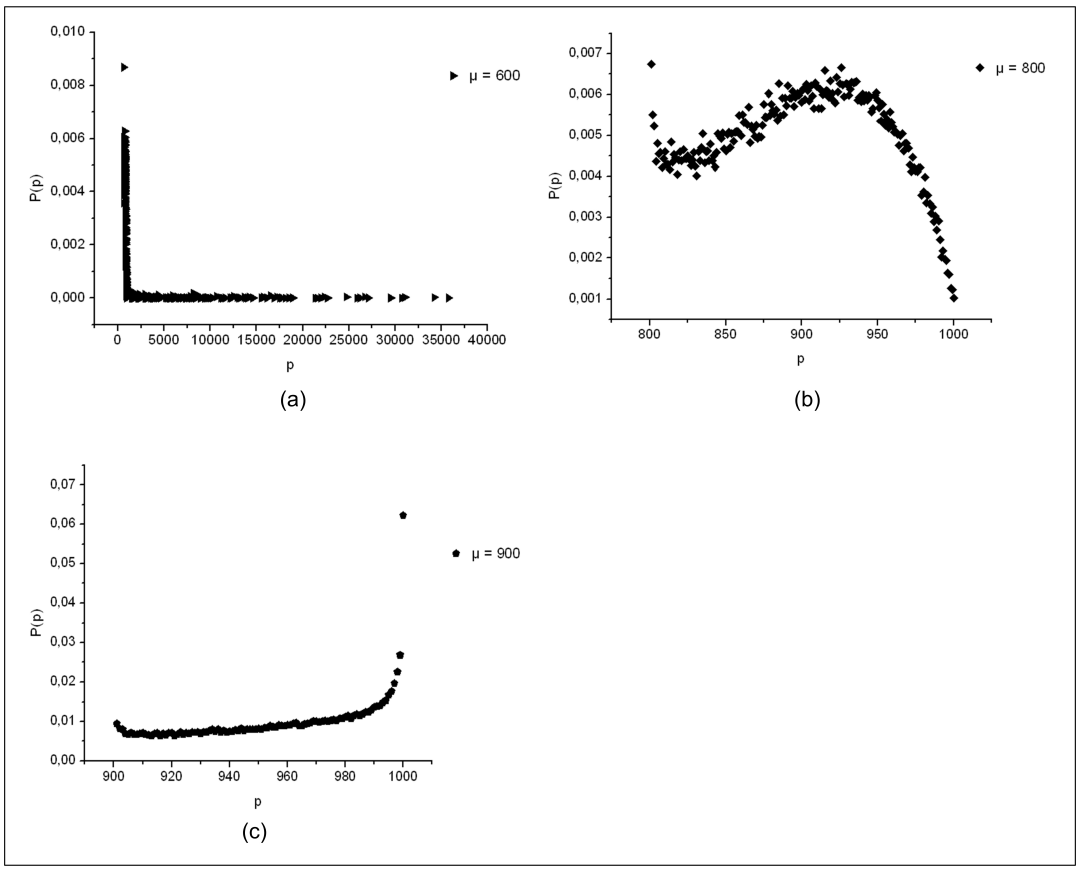


Figura 5.5: Distribuição de período de atratores obtidos a partir de simulações com $M = 1000$, $N = 1000$ e μ variando da seguintes forma: a) $\mu = 600$, b) $\mu = 800$ e c) $\mu = 900$.

Fig. 5.7, que apresenta a evolução de n_s em função de p, para $N = 1000$ e diferentes valores de μ.

Nos gráficos apresentados na Fig. 5.7, podemos perceber que ao ampliarmos o valor de μ, aumenta-se também a frequência de atratores do tipo $p \neq n_s$, culminando em atratores, nos quais $p \gg N$, ao utilizarmos $\mu = 600$, como apresentado na Fig. 5.6. A partir desse valor de μ, obtemos a zona de convergência, em que ainda ocorrem atratores do tipo $p \neq n_s$, porém com frequências cada vez menores, até obtermos, a partir de $\mu = 800$, apenas atratores em que $p = n_s$. Esse comportamento, juntamente com os demais comportamentos explorados anteriormente, permite-nos entender o aumento da frequência de períodos do tipo $p = N$, conforme destacado na Fig. 5.4.

5.2.3 Probabilidade Conjunta de Tempos de Transiente e Período de Atratores

Na Fig. 5.8 é possível visualizar a distribuição de probabilidade conjunta de tempos de transiente e período de atratores para o espaço bidimensional, para $N = 1000$. Os resultados

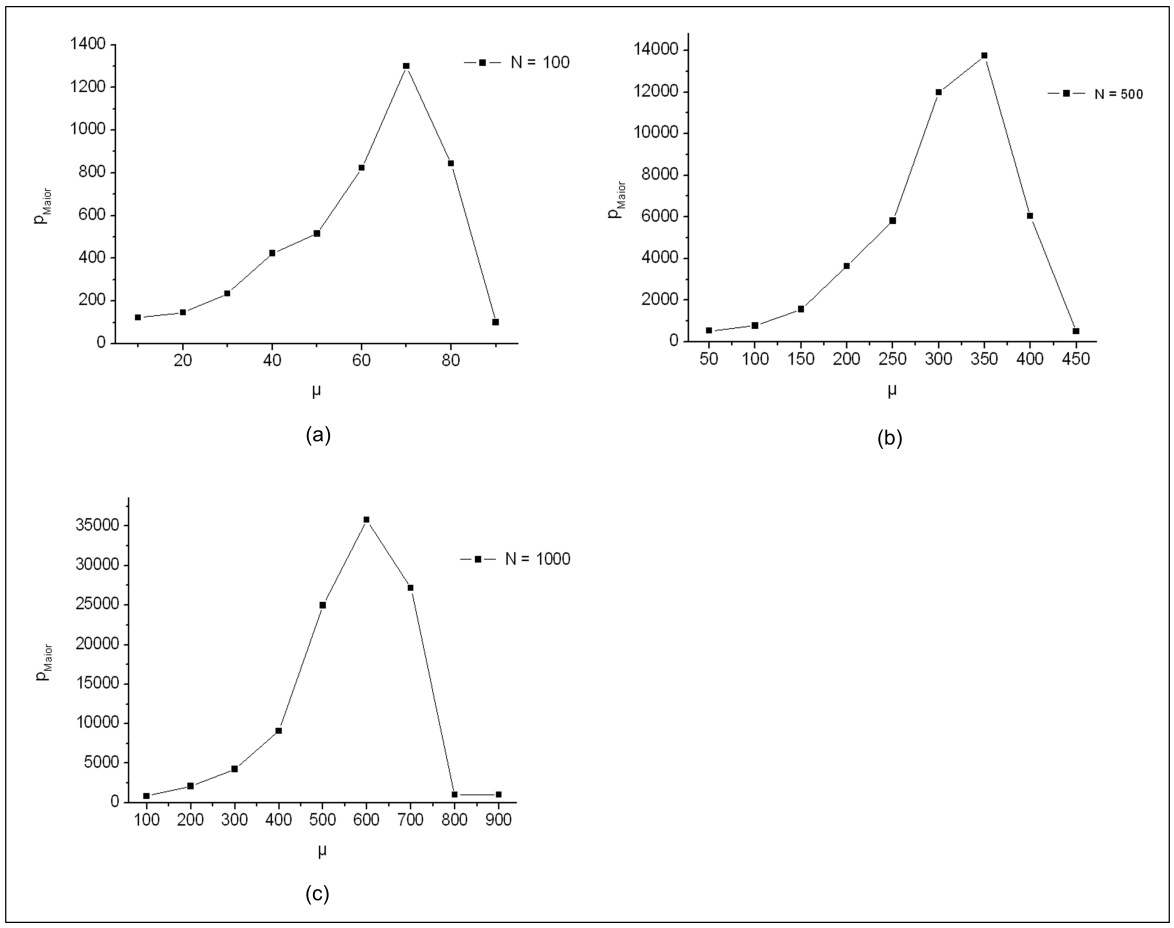


Figura 5.6: Maiores períodos registrados para diferentes combinações de N e μ. Foram utilizados $M = 1000$ realizações, sendo que em a) $N = 100$, b) $N = 500$ e $N = 1000$.

foram obtidos com $M = 1000$ realizações, sendo adotados diferentes valores de memória ($25 \leq \mu \leq 900$) a fim de explorar o comportamento do turista com os mais variados conhecimentos sobre o meio.

Como podemos perceber, ao utilizar $\mu = 25$, o que representa uma vizinhança de 25% do ambiente, a maioria das caminhadas são formadas por transientes e períodos pequenos, destacando-se caminhadas em que o transiente é nulo e os atratores com $26 \leq p \leq 36$. Porém, observamos que, a medida em que o valor do transiente aumenta, a ocorrência de atratores com $100 \leq p \leq 400$ passam a ocorrer de maneira uniforme.

Ao ampliarmos a memória para $\mu = 100$, percebemos que aumenta a probabilidade de ocorrência de atratores com $101 \leq p \leq 400$ se comparada ao caso anterior. Todavia, a distribuição de atratores mantém o comportamento do cenário anterior, em que atratores pequenos são mais frequentes. Percebemos também que, à medida que analisamos transientes com $t \approx 1000$, a frequência de atratores de períodos elevados diminui.

Quando utilizamos $300 \leq \mu \leq 500$, concluímos que ocorrem os maiores tempos de tran-

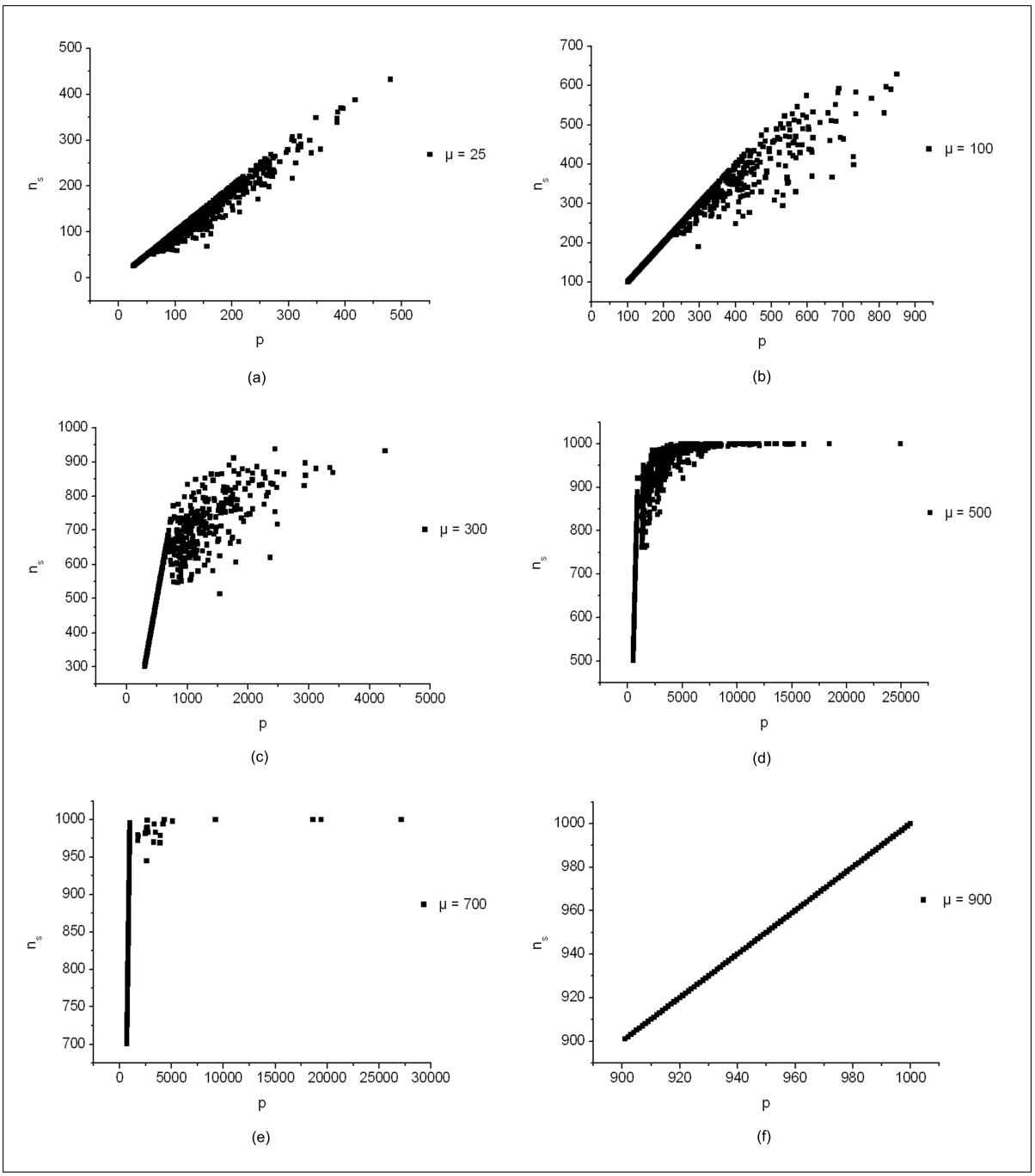


Figura 5.7: Evolução de n_s em função de p para $N = 1000$ e $M = 1000$. a) $\mu = 25$, b) $\mu = 100$, c) $\mu = 300$, d) $\mu = 500$, e)$\mu = 700$ e f) $\mu = 900$.

sientes e de período de atratores. Neste cenário, apesar da maior frequência de tempos de transiente e de período de atratores, ambos pequenos, percebemos que à medida em que os tempos de transientes aumentam, as frequências de tempos de transientes e períodos de atratores passam a ser igualmente prováveis.

Ao ampliarmos ainda mais o valor da memória utilizada, fazendo $700 \leq \mu \leq 900$, notamos que, ainda assim, a frequência de tempos pequenos de transientes, juntamente com período de

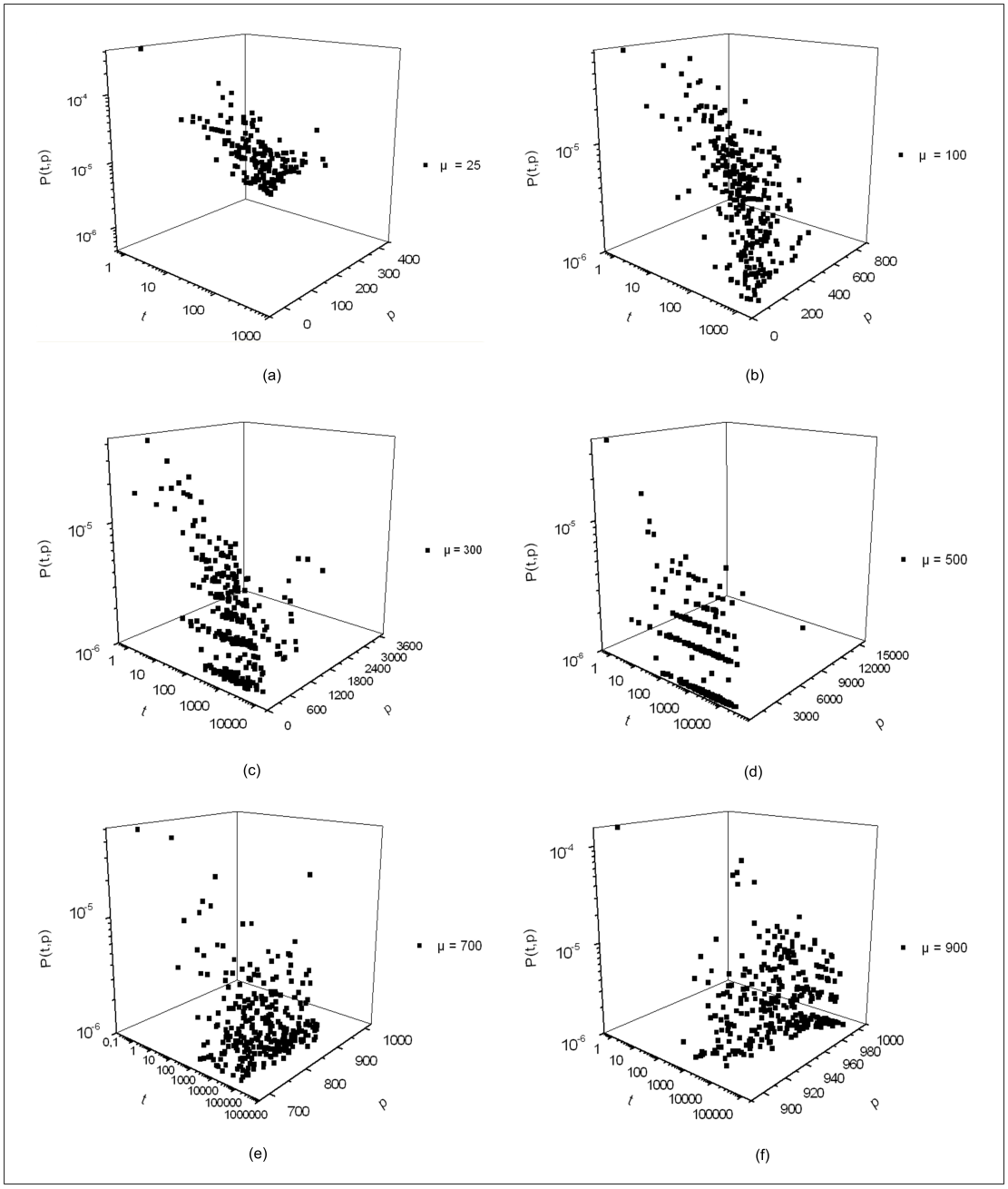


Figura 5.8: Distribuição de probabilidade conjunta de tempos de transientes e período de atratores para o espaço bidimensional com $N = 1000$ e $M = 1000$. Foram adotados diferentes valores de μ, sendo que: a) $\mu = 25$, b) $\mu = 100$, c) $\mu = 300$, d) $\mu = 500$, e) $\mu = 700$ e f) $\mu = 900$.

atratores mínimos, ainda são mais elevadas. Todavia, percebemos também uma elevação na ocorrência de atratores onde $p \approx N$. Porém, as caminhadas que possuem tais atratores ainda são formadas por tempos de transientes pequenos. Essa situação reflete aquela descrita nas Fig. 5.1 e 5.4, onde os tempos de transientes elevados são mais raros que aqueles de tempos pequenos, ao contrário do período de atratores quando μ tende a N, onde os períodos de $O(N)$ são mais frequentes.

5.3 Resultados para modelos $d \geq 3$

Nesta seção apresentamos os resultados obtidos nos espaços com $d = 3$, $d = 20$ e $d = 50$. Também realizamos estudos com o espaço $d = 10$. Todavia, devido às semelhanças encontradas nos resultados, não se faz necessário exibi-los.

Ao realizar os estudos da caminhada do turista nos espaços dimensionais anteriormente citados, procuramos ampliar o espaço amostral considerado, utilizando $100 \leq N \leq 10.000$, a fim de verificar sua influência na caminhada do turista. Desta forma, valores elevados para a memória também são necessários, o que nos levou à considerar $2 \leq \mu \leq 400$.

Durante os estudos realizados, ao obtermos um conjunto de resultados suficiente, que permitiu analisar o comportamento do turista em espaços com $d \geq 3$, percebemos que, assim como no espaço bidimensional, as distribuições analisadas não são influenciadas unicamente pela ampliação (ou redução) na quantidade de sítios por mapa, mas sim pelo produto $N \times \mu$. Desta forma, sempre que precisarmos demostrar tal relacionamento, onde $\mu \approx N$, optamos por utilizar mapas com $N = 100$. Esta medida foi adotada pois, com a solução do problema da quantidade de memória RAM ocupada, através da estrutura HL, ainda existem restrições computacionais para se explorar caminhadas cujos fatores possuem grandezas elevadas.

Tais restrições não impediram o estudo dos sistemas em questão, mas dificultam as simulações numéricas. Dentre tais restrições, encontra-se o tempo de processamento, que aumenta exponencialmente, á medida em que os fatores M, N e μ aumentam linearmente.

Sabemos que nas arquiteturas atualmente utilizadas, os processadores estão chegando aos limites impostos pela natureza, principalmente pelas leis físicas da velocidade da luz e ainda, aquelas que descrevem a dissipação do calor. Desta forma, as gerações de processadores atualmente comercializados são desenvolvidos com múltiplos núcleos de processamento em apenas uma pastilha. Todavia, apesar de haver mais núcleos de processamento, estes elementos possuem, em geral, taxas de *clock*[1] inferiores aos processadores mononucleados.

A existência de múltiplos núcleos de processamento permite a execução de múltiplas tarefas utilizando paralelismo real, e não apenas a programação concorrente, em que há um pseudo-paralelismo envolvido, em que apenas um núcleo simula a existência de múltiplos, como as máquinas com suporte SMP (Symmetric Multiprocessing), ou ainda, como a preempção de tarefas realizadas pelos sistemas operacionais. Neste caso, o pseudo-paralelismo é obtido em nível de *software*. Apesar dessa possibilidade, os sistemas operacionais ainda não estão preparados

[1] Taxa de *Clock* representa a quantidade de operações que um processador pode executar por unidade de tempo. Sua medida dá-se em *Hertz*.

para usufruir de vários núcleos para a execução de um único processo, sendo assim, a execução de simulações, com valores elevados de mapas, sítios e memória, possuem tempos de execução elevados, retardando a obtenção dos resultados. Vale lembrar que essa limitação pode ser tratada com programação paralela, através dos paradigmas de passagem de mensagens e *threads*[2], onde obtemos paralelismo inter e intra processos.

5.3.1 Distribuição de Tempos de Transiente

A distribuição de tempos de transiente, para o espaço tridimensional, pode ser observada na Fig. 5.9, cujos valores de $N \in \{100, 1000, 2000, 10000\}$.

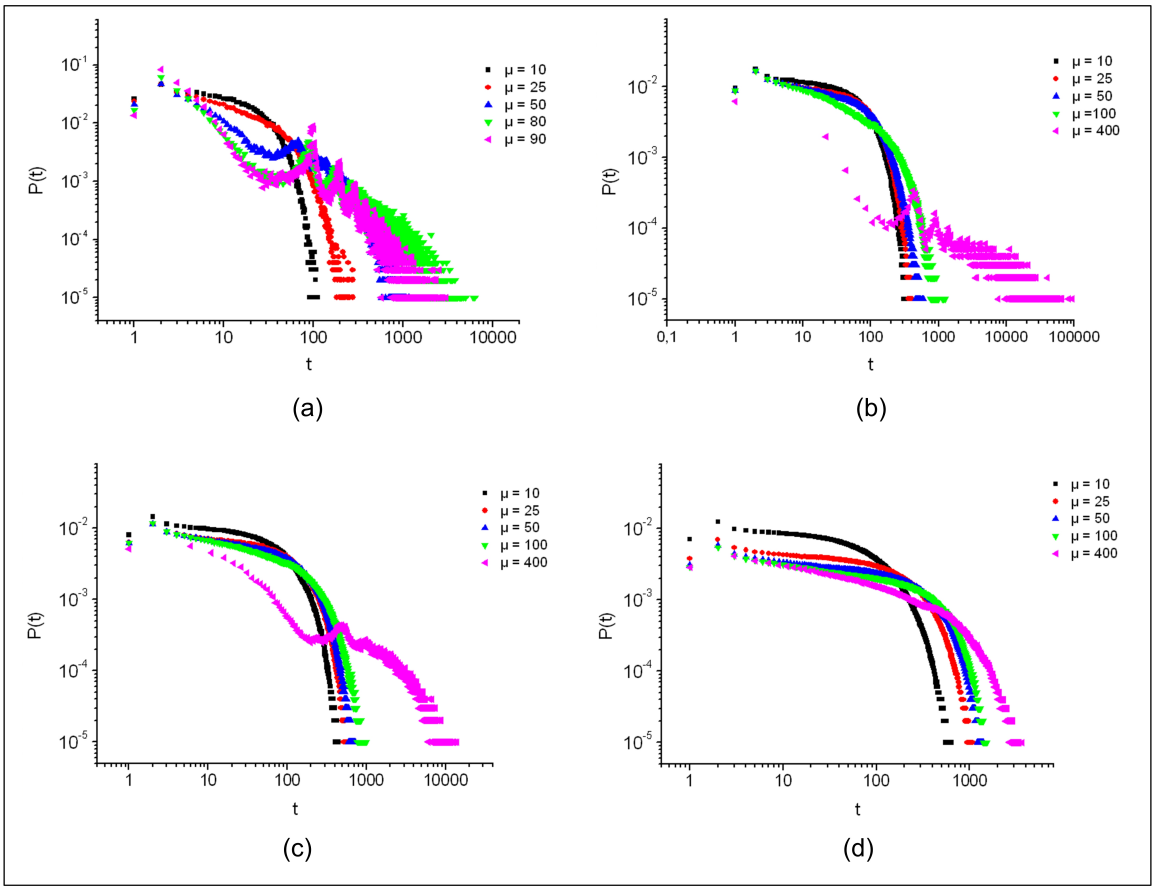


Figura 5.9: Distribuição de tempos de transiente no espaço tridimensional, considerando $M = 1000$ realizações, sendo que: a) $N = 100$, b) $N = 1000$, c) $N = 2000$ e d) $N = 10000$.

Como podemos observar, ao compararmos os resultados obtidos na Fig. 5.9.a e aqueles obtidos na Fig. 5.3, que contém os resultados para $N = 100$ em $d = 2$, percebemos que as distribuições de probabilidade, de ambos, são muito próximas, onde transientes pequenos são

[2] *Threads* são linhas de execução em um processo interno ao sistema operacional, assim, todas as *threads* de um processo compartilham seu espaço de endereçamento, podendo trocar informações via memória compartilhada.

obtidos com maior frequência, sendo que para $\mu \leq 10$, as curvas obtidas podem ser representadas por uma equação exponencial. Porém, a partir desse ponto, é possível observarmos que alguns conjuntos de transientes possuem frequências mais elevadas que seus antecessores e sucessores.

É possível observarmos, ainda, que no espaço tridimensional, para um dado μ, os tempos de transientes são maiores que aqueles obtidos no espaço bidimensional, o que explica os decaimentos mais acentuados nas distribuições que compõem os resultados obtidos no espaço tridimensional.

Ao ampliarmos a quantidade de sítios por mapa, é possível notar que o comportamento das distribuições dos tempos de transientes se mantém, o que demonstra que elas não dependem unicamente de N, como mencionado no início deste capítulo.

Tal comportamento também é obtido nas distribuições de tempos de transientes para os espaços amostrais $d > 3$. Na Fig. 5.10 é possível observarmos os resultados para os espaços $d = 20$ e $d = 50$, ao realizarmos experimentos com $2 \leq \mu \leq 90$ e $N = 100$.

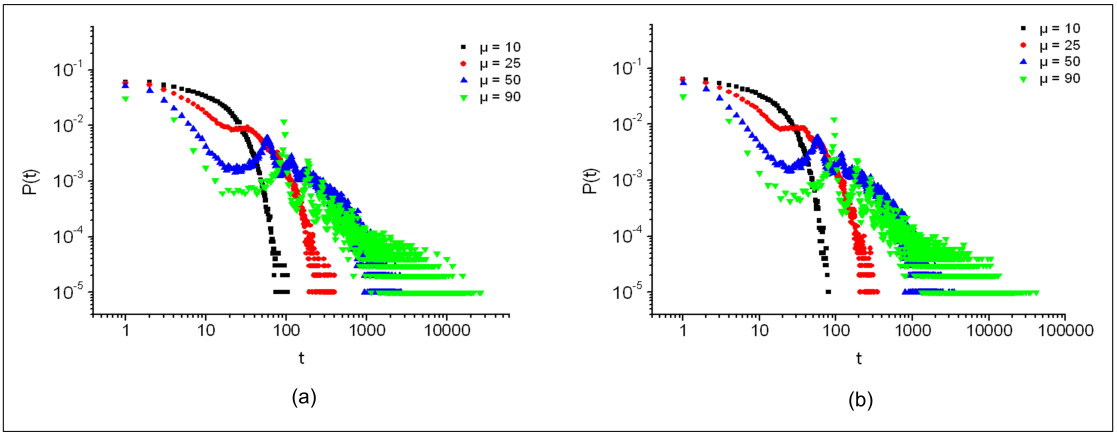


Figura 5.10: Distribuição de tempos de transiente nos espaços dimensionais (a) $d = 20$ e (b) $d = 50$, considerando $M = 1000$ realizações e $N = 100$ sítios.

Ao analisarmos as distribuições obtidas, percebemos que, à medida em que aumentamos a dimensão considerada, há um aumento nos tempos de transientes obtidos. Quando $d = 50$ e $\mu = 90$, foram observados tempos de transientes, aproximadamente, dez vezes maiores que aqueles obtidos no espaço tridimensional. Tal característica justifica a redução das frequências dos tempos de transientes, à medida que ampliamos o espaço dimensional, uma vez que a quantidade de caminhadas, $M \times N$, continua inalterada. A ampliação nos tempos de transientes é causada pela ampliação das possibilidades de movimentação do turista em um dado sítio.

5.3.2 Distribuição de Período de Atratores

A distribuição de período de atratores, para o espaço tridimensional, pode ser observada na Fig. 5.11, onde $N \in \{100, 1000, 2000, 10000\}$, seguindo o modelo proposto no ínicio da seção.

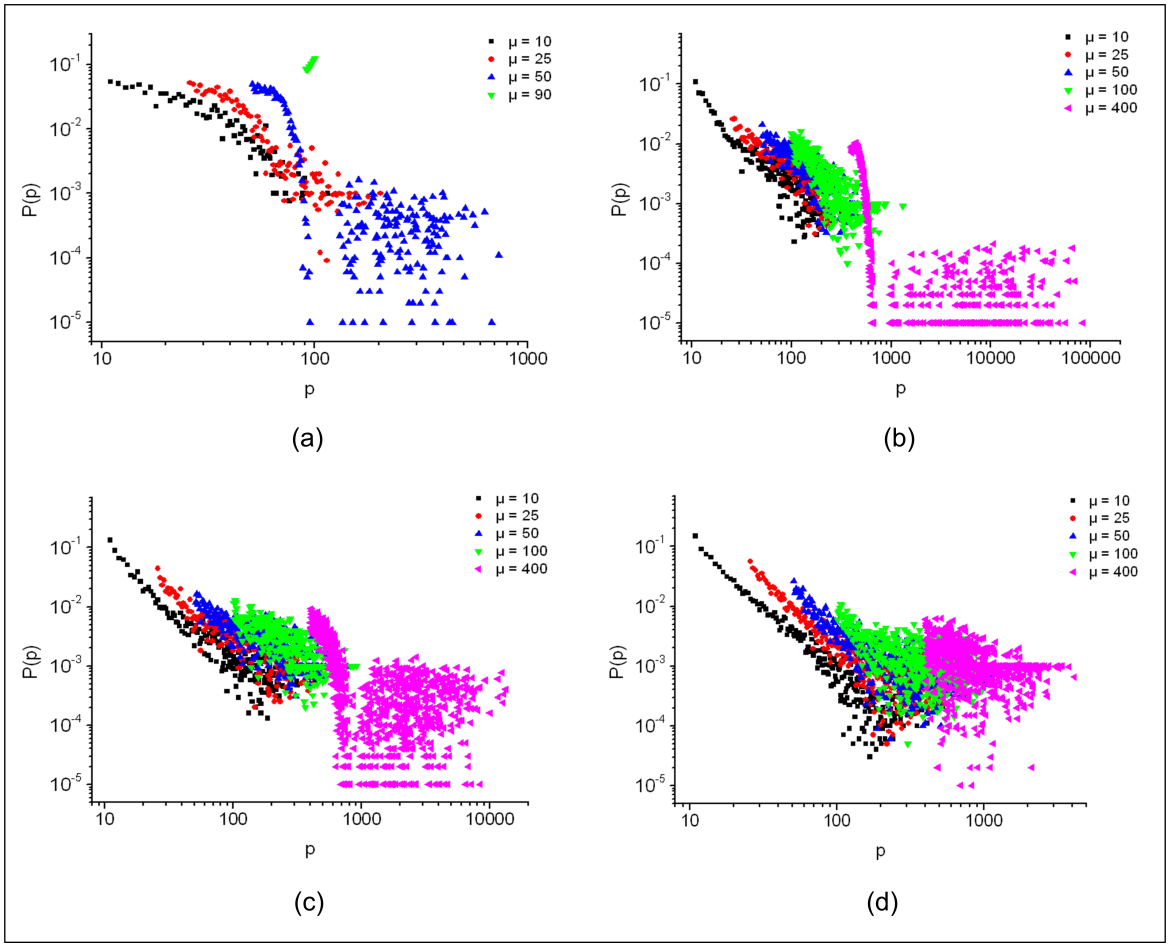


Figura 5.11: Distribuição de período de atratores no espaço tridimensional, considerando $M = 1000$ realizações, sendo que: a) $N = 100$, b) $N = 1000$, c) $N = 2000$ e d) $N = 10000$.

Analisando a Fig. 5.11, percebemos que o comportamento, das distribuições obtidas, são semelhantes aos observados no espaço bidimensional, apresentado na Fig. 5.4. Todavia, ao analisarmos os resultados obtidos com $d = 2$ e $N = 100$, percebemos que há uma redução na frequência de alguns períodos de atratores enquanto que, em outros, há um aumento. Tal fato pode ser melhor observado ao compararmos as distribuições obtidas no espaço tridimensional com aquelas obtidas nos espaços $d = 20$ e $d = 50$, cujas distribuições podem ser observadas na Fig. 5.12.

Um fator importante a ser observado refere-se a ausência na inversão de frequência de período de atratores para valores elevados de μ. Podemos observar esse comportamento ao considerarmos $\mu = 90$. Tanto no espaço bidimensional, quanto no tridimensional, há uma inversão,

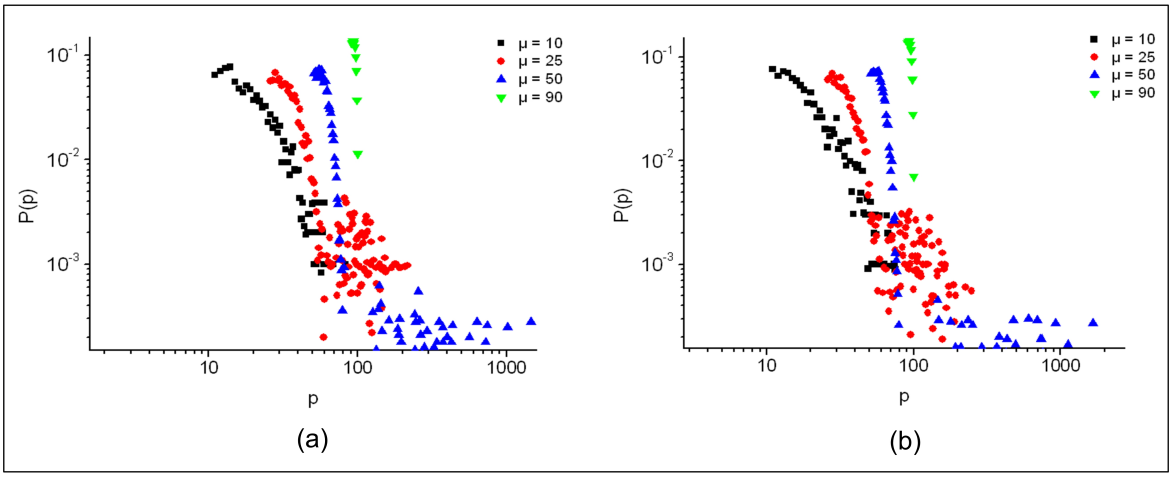


Figura 5.12: Distribuição de período de atratores nos espaços dimensionais $d = 20$ e $d = 50$, considerando $M = 1000$, sendo que: a) $d = 20$ e b) $d = 50$.

onde os atratores de períodos elevados passam a ocorrer mais frequentemente que os menores. Tal fato é observado quando $\mu \approx N$. Contudo, para espaços dimensionais $d > 3$, essa inversão não ocorre, uma vez que, mesmo adotando $\mu \approx N$, a frequência de períodos do tipo $\mu + 1$ são maiores que aqueles do tipo $p = N$. Para tal estudo, consideramos $N \in \{100, 1000, 2000\}$ e $2 \leq \mu \leq N - 1$.

5.3.3 Probabilidade Conjunta de Tempos de Transiente e Período de Atratores

Da mesma forma que ocorreu nas distribuições apresentadas anteriormente, as distribuições de probabilidade conjunta de tempos de transiente e períodos de atratores, nos espaços dimensionais $d \geq 3$, apresentam comportamento semelhante aos obtidos no espaço bidimensional. Tais evidências podem ser observadas na Fig. 5.13, que apresenta as distribuições em questão, envolvendo $N = 100$, $M = 1000$ e $25 \leq \mu \leq 90$.

Analisando a Fig. 5.13, percebemos que, independente das configurações utilizadas para N e μ, as maiores frequencias são registradas com pequenos valores de tempos de transientes e de período de atratores. Todavia, ao aumentarmos o valor da memória utilizada, tal que $\mu \approx N$, a ocorrência de períodos elevados tornam-se maiores, bem como os tempos de transientes.

Para os espaços dimensionais $d > 3$, os gráficos obtidos apresentam o mesmo comportamento que aquele retratado na Fig. 5.13, confirmando nossa expectativa, de que, a partir do espaço tridimensional, o fator dimensionalidade não afeta significativamente o comportamento do sistema.

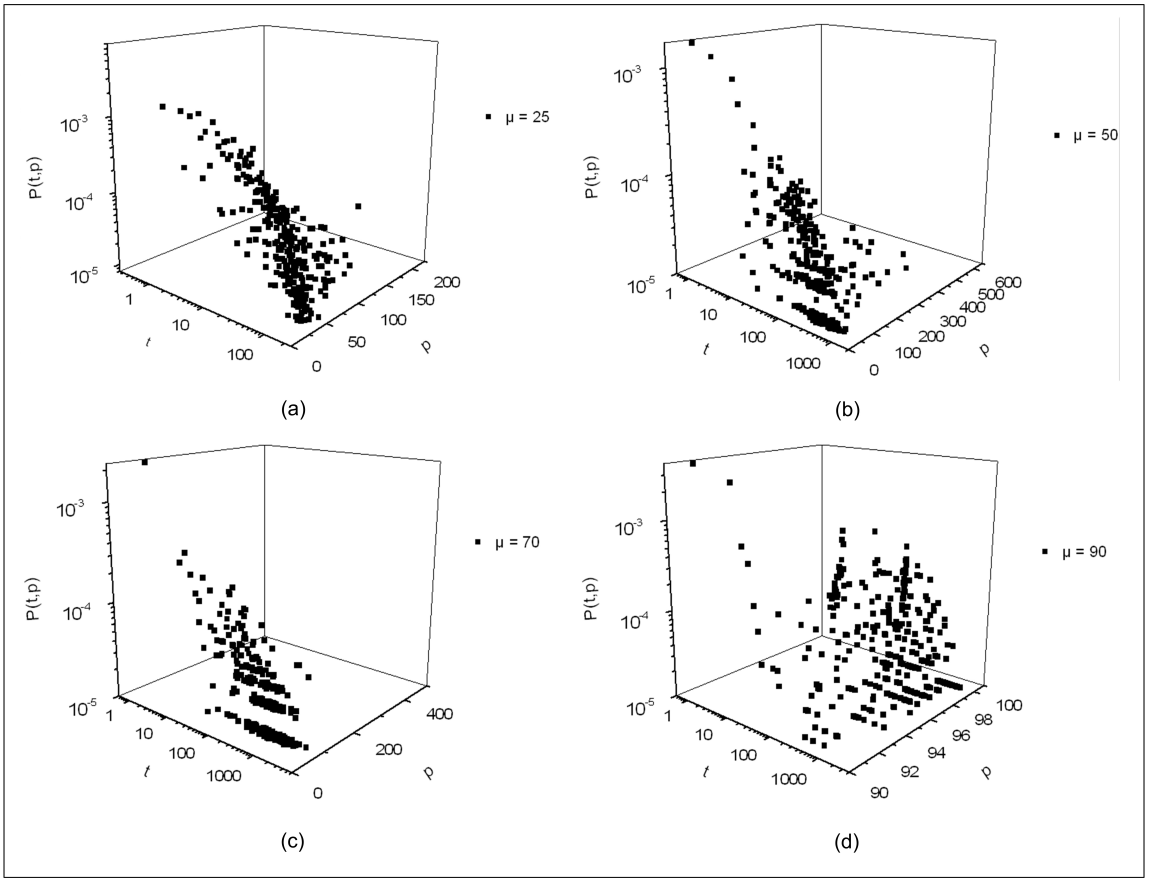


Figura 5.13: Distribuição conjunta de tempos de transiente e período de atratores no espaço tridimensional considerando $N = 100$ e $M = 1000$, onde: a) $\mu = 25$, b) $\mu = 50$, c) $\mu = 70$ e d) $\mu = 90$.

5.4 Conclusão

Neste Capítulo pudemos demonstrar que foi possível, através do uso de simulações numéricas, avaliarmos o comportamento determinista da caminhada do turista, utilizando um valor de memória μ da ordem de N. Esses resultados foram obtidos a partir do desenvolvimento da estrutura de lista HL, implementada no modelo numérico do sistema. O nosso objetivo foi o de analisar as distribuições estatísticas geradas e cotejar ao comportamento determinista do turista, anteriormente observado em trabalhos onde a dinâmica ocorria em sistemas menores.

Inicialmente analisamos a distribuição de probabilidade de tempos de transientes, onde percebemos a existência de uma fase estacionária entre dois regimes de ampliação da exploração do meio. Nesta fase, à medida que μ é incrementado, a quantidade de sítios explorados apresenta pouca variação, diferentemente das fases anterior e posterior a esse momento.

Já ao considerarmos as distribuições de probabilidade de período de atratores, percebemos, principalmente, a existência de três comportamentos distintos. Em um primeiro momento, a

presença de atratores do tipo $p = \mu + 1$ é mais freqüente, diferentemente do segundo em que formações do tipo $\mu + 1 \leq p \leq N$ também são estão presentes em grande quantidade. É nessa fase que encontramos a formação dos maiores períodos em que $p \gg N$. Já em um terceiro momento, percebemos que os períodos de atratores tendem a ser limitados pela quantidade de sítios do mapa, assim a formação $p = N$ é mais freqüente. Tais fatos foram comprovados ao analisarmos as distribuições de probabilidade conjunta de período de atratores e n_s.

Também para o espaço bidimensional, apresentamos a distribuição de probabilidade conjunta de tempos de transientes e de períodos de atratores, demonstrando, dentre outros comportamentos, que a formação tanto de transientes, quanto de períodos pequenos são mais freqüentes independente da configuração de N e μ.

Finalmente, analisamos os espaços $d = 3, d = 20$ e $d = 50$, onde comprovamos que o comportamento do caminhante para esses espaços dimensionais é semelhante ao obtido no espaço bidimensional, apresentando apenas alterações de grandeza dos dados obtidos, caracterizando assim uma transição de fase entre regimes de exploração "curto" e "longo" ao considerarmos sistemas de baixa e alta dimensionalidade.

6 Conclusão

Nesse trabalho de pesquisa, procuramos estudar, observar e ver o alcance da dinâmica determinista do turista, ao submetermos a memória μ do camihante na ordem de N, no espaço $d > 1$. Através do uso do modelo numérico da caminhada, modificado em virtude do objetivo esperado para esta tese, analisou-se vários cenários onde pode-se vislumbrar uma dinâmica determinista do turista, em sistemas muito maiores dos que já se tinham investigado. E, independentes do espaço onde se apresenta a dinâmica do turista, na caminhada determinista, as trajetórias sempre terminam em ciclos, os quais apresentam uma distribuição nada trivial de períodos.

Em um dos casos mais comuns do modelo, porém, não menos exclusivo que os demais casos do turista, quando o caminhante tem que se movimentar, indo para a cidade mais próxima, com memória $\mu = 1$, a distribuição dos períodos é igual a $P_{1,d}(p) = \delta_{p,2}$, ou seja, todo período é formado por um casal. Notou-se que a medida que consideramos valores maiores de memória, cada caminhada se torna parcialmente autorepulsiva e o menor período permitido para um ciclo de atrator é $p = \mu + 1$, enquanto que o maior período possível não pudemos definir analiticamente. Esta previsão é válida tanto para o caso unidimensional, quanto para dimensões mais altas, e uma vez que pudemos contemplar esses casos no espaço d dimensional, verificamos que ao empregarmos uma memória muito pequena, da ordem de $\mu \ll N$, o caminhante é capaz de atravessar todo o meio, devido a uma mudança de regimes, encontrados na exploração do turista, provocada a partir de um ponto crítico. Entretanto, isso ocorre apenas no espaço unidimensional, no modelo determinista com memória, no meio com condições aberta ou fechada de contorno.

Essa mesma transição de regimes, a qual nos referimos anteriormente, não ocorre em dimensões maiores, devido ao fato de que, em dimensões mais altas, a homogeneidade de comportamentos predominante na dinâmica do turista, não nos permite vislumbrar uma exploração abrupta do meio, quando varia-se a memória do caminhante de $2 \leq \mu \leq N$. Porém, no caso $d > 1$, notamos uma diferença de comportamento com relação à distribuição de probabilidades das grandezas t e p, na caminhada determninista do turista, em função de um aumento significativo da memória.

Depois de apresentarmos uma revisão sobre o modelo de caminhadas deterministas do turista, abordando as características do meio aleatório em que essas caminhadas ocorrem, suas estatísticas de vizinhança, bem como, o desenvolvimento da dinâmica no sistema, podemos concluir que a proposta de alterarmos a estrutura de armazenamento de dados no modelo numérico da caminhada, nos proporcionou um saldo muito positivo, no consumo da RAM. Dessa forma pudemos enxergar o que a dinâmica do turista nos reservava mais além.

Do contrário, quando buscávamos estender a memória μ do caminhante até o valor de N, usando o modelo com a estrutura de dados do tipo matriz, este consumia quantidades de memória RAM, maiores que a disponível fisicamente no equipamento utilizado, e quando esse fato ocorria, o módulo de gerenciamento de processos, enviava uma sinal, do tipo *sigkill* para o sistema operacional, ordenando que finalizasse a simulação, não prejudicando o sistema computacional, mas, nos impedindo de defrontar com o comportamento da caminhada com grandezas tão elevadas.

Também, com a inserção de estruturas do tipo listas, no modelo numérico, pode-se fixar mais uma grandeza no sistema, com a qual foi possível visualizar a composição dos pontos de um ciclo de atrator, compreendendo onde recomençam a formação de ciclos semelhantes à polígonos complexos (ou laços), e demais características, oriundas, desse comportamento. E, auxiliados com o simulador visual de trajetórias, explicamos o aparecimento de regiões proibidas do mapa, no modelo unidimensional, onde ciclos de atratores nunca se formam, devido a imposição da dinâmica determinista, agregados à geometria do meio aleatório. E uma vez verificada a faixa de atratores proibidos,

Também afixando valores muito altos de memória, no modelo do turista com memória, descobrimos que uma caminhada, ao contrário do que se previa, pode gerar trajetórias, cujos períodos chegam a proporções muito maiores que N, ou seja, $p \gg N$. No entanto, à medida que μ tende a N, percebemos que os períodos de atratores formados são do tipo $p = N$, definindo assim comportamentos distintos à medida que ampliamos a memória μ do sistema.

Comportamentos distintos também foram observados na exploração do meio $(t + n_s)$, ao considerarmos valores altos de memória. Observamos a formação de uma fase estacionária e intermediária, em que a exploração do meio estabiliza-se, diferentemente dos momentos anterior e posterior a essa fase, em que a exploração do meio aumenta à medida que aumentamos o valor de μ.

A partir da possibilidade de estudar sistemas de altas dimensionalidades, descobrimos que as características das caminhadas para espaços dimensionais $d \geq 2$ mantém-se inalteradas, apresentando apenas incrementos de valores, o que nos permitiu concluir que trajetórias extrema-

mente longas são características de sistemas de alta dimensionalidade, uma vez que ao ampliarmos a dimensionalidade, cada ponto ganha mais "liberdade", uma vez que há mais direções de saída para o caminhante. Este fato caracteriza uma transição de regimes de exploração "curto" e "longo". No entanto, novos estudos sobre esse comportamento deverão ser realizados. O que nos permite propor a seguir novos trabalhos, que poderão ser desenvolvidos futuramente.

6.0.1 Trabalhos futuros

Devido à amplitude da área de pesquisa em que este trabalho se encontra inserido, novos trabalhos são necessários para ampliarmos o conhecimento da caminhada do turista. Dentre os mais importantes destacam-se:

- Apesar do grande avanço obtido com a estrutura HL, onde sistemas de alta dimensionalidade e com valores elevados de μ passaram a ser passíveis de exploração, há ainda problemas relacionados ao desempenho do código de simulação, uma vez que em sistemas com essas características, as caminhadas geradas são longas, consumindo tempos elevados de processamento. Sendo assim, não é incomum aguardar dias e em alguns casos semanas para obter certos resultados.

  Desta forma, propõe-se a paralelização do código de simulação, utilizando para tal sistemas de passagem de mensagens como MPI (*Message Passing Interface*). Na análise da paralelização pode-se considerar cada caminhada como um item independente, desta forma, deve-se enviar um conjunto de caminhadas a um nó do *cluster* a fim de que sejam executadas. No entanto, deve-se observar que mais de uma caminhada deve ser enviada por vez a um nó, caso contrário, se o envio for unitário, há a possibilidade de congestionarmos o meio de comunicação, devido à grande quantidade de mensagens trocadas.

- Tratando ainda de implementação das soluções de *software* desenvolvidas neste trabalho, propõe-se o desenvolvimento das classes *Visualizador_2D* e *Visualizador_3D*, completando assim o projeto do aplicativo Turista JVisual. Essa implementação permitirá estudar características envolvidas na formação das caminhadas, tanto de transientes quanto de atratores.

- Uma vez que detectamos as semelhanças entre os resultados obtidos no hiperespaço, faz-se necessário novos estudos para investigar a transição de fases detectada entre regimes de exploração "curto" e "longo", destacada no Capítulo 5. Isso permitirá uma melhor caracterização do estudo do *hiperespaço*.

- Ainda baseado nos estudos do *hiperespaço*, propõe-se desenvolver estudos comparativos entre os resultados obtidos neste trabalho, com aqueles obtidos utilizando o método *Random Link*. Este se apresenta como o limite de dimensão infinita do espaço euclidiano. Ao simularmos sistemas utilizando o método *Random Link*, algumas configurações levam a finalização abrupta do programa de simulação, no entanto, não foi possível detectar se essa característica deve-se a obtenção de um atrator de período grande, ou se a formação de atratores não ocorre nessas configurações, apresentando-se assim como o caos determinista.

  Desta forma, acreditamos ser possível definir tal característica através do programa de simulação utilizado neste trabalho, utilizando dimensões extremamente elevadas. No entanto, para tal se faz necessário a paralelização do código de simulação.

- A partir da determinação da fase estacionária obtida na exploração do meio no hiperespaço, propõe-se desenvolver novos estudos a fim de determinar limites, se possível, para as mudanças, definindo assim uma relação entre as grandezas μ, N e a dimensionalidade considerada.

APÊNDICE A – Implementação da estrutura HL

A implementação do tipo abstrato de dados HL foi realizado na linguagem de programação C. Desta forma, atendendo aos requisitos de modularidade disponbilizado por essa linguagem, bem como aos príncipios de qualidade de *software*, a implementação está dividida em duas partes. A primeira, denominada *cabeçalho* e contida no arquivo hl.h, contém a declaração do tipo abstrato HL, bem como o cabeçalho das funções que a manipulam, caracterizando assim uma espécie de *interface*. Já a segunda parte, denominada *implementação* e localizada no arquivo hl.c, contém a implementação das funções definidas na *interface*.

A.0.2 Definindo o tipo HL

A seguir é apresentado o código de definição da estrutura de dados HL. Como descrito anteriormente, o código apresentado de ser inserido no arquivo de cabeçalho.

É importante ressaltar, que comentários extras inseridos no arquivo original foram suprimidos, no entanto, o código aqui contido contém todas as partes necessárias para o seu correto funcionamento.

```
#ifndef ED_H
#define ED_H

/* ====================================== */
/* === Declarando a lista de ocorrencias  */
/* ====================================== */
typedef struct TElementos *TLElementos;

struct TElementos {
    long int elementos;
    long int contElementos;
    TLElementos prox;
};

/* ====================================== */
```

```
/* === Declarando a lista de periodos     */
/* ====================================== */
typedef struct TPeriodo *TLPeriodo;

struct TPeriodo {
    long int periodo;
    long int contDistintos;
    long int contRepetidos;
    TLPeriodo prox;
    TLElementos LElementos;
};

/* ====================================== */
/* === Declarando a lista de transientes  */
/* ====================================== */
typedef struct TTransiente *TLTransiente;

struct TTransiente {
    long int trans;
    long int contTrans;
    TLTransiente prox;
    TLPeriodo LPeriodo;
};
```

Após realizar a declaração da estrutura, denominada TLTransiente, faz-se necessário definir as suas funções. Desta forma, o código necessário para tal será apresentado a seguir, juntamente com os comentários pertinentes.

```
/* Inicializa a lista de transientes com o valor NULL */
void InicializarLT(TLTransiente *);

/* Acrescenta na lista de transientes uma ocorrência formada por:
   1) Transiente,
   2) período e
   3) ns */
int AcrescentarLT(TLTransiente *, long int, long int, long int);


/* Acrescenta na lista de periodos uma ocorrência formada por:
   1) Periodo e
   2) ns */
int AcrescentarP(TLPeriodo *, long int, long int);

/* Acrescenta na lista de elementos uma ocorrência, tendo como
   parametro a quantidade de elementos que formaram um periodo. */
int AcrescentarE(TLElementos *, long int);

/* Retorna a quantidade de vezes que um transiente t ocorreu
   com um determinado periodo p. */
long int QuantTransPeriodo(TLTransiente, long int, long int);
```

```
/* Recebe como parâmetro uma lista de transientes (TLTrasiente),
   retornando uma lista de periodos (TLPeriodo), que acumula,
   em cada elemento da lista, todas as ocorrências de periodos nos
   mais diversos transientes. */
int UnificarP(TLTransiente, TLPeriodo *);

/* Desaloca lista de transiente (TLTransiente), liberando a memória por ela ocupada. */
void DestruirLT(TLTransiente *);

/* Desaloca a lista de periodos (TLPeriodo). */
void DestruirLP(TLPeriodo *);

/* Desaloca a lista de elementos (TLElementos). */
void DestruirLE(TLElementos *);

#endif
```

A.0.3 Implementando as funções sobre HL

A seguir encontra-se o código de implementação das funções declaradas no arquivo de cabeçalho e listas na Seção A.0.2. Como poderá ser observado, a ligação entre a implementação e a definição do tipo, e de suas funções, dá por uma cláusula *include*.

```
#include <stdio.h>
#include "ed.h"


void InicializarLT (TLTransiente *l)
{
    *l = NULL;
};

/* ================================================================================ */
int AcrescentarLT(TLTransiente *l, long int transiente, long int periodo, long int elementos)
{
    if (*l == NULL){
    /* === Inserindo no início da lista === */
        if ((*l = (struct TTransiente *) malloc(sizeof(struct TTransiente))) == NULL)
            return 0;

        (*l)->trans = transiente;
        (*l)->contTrans = 1;
        (*l)->prox = NULL;
        (*l)->LPeriodo = NULL;
        if (AcrescentarP(&((*l)->LPeriodo), periodo, elementos))
            return 1;
        else
            return 0;
    }
    else {
```

```
    /* === Variáveis auxiliares === */
        TLTransiente aux,aux2, anterior;

        anterior = NULL;
        aux = *l;
        while (((aux) != NULL) && (aux->trans < transiente)) {
            anterior = aux;
            aux = aux->prox;
        }
        if ((aux != NULL) && (aux->trans == transiente)){
            /* === Transiente encontrado, incrementando contador === */
            aux->contTrans++;
            if (AcrescentarP(&(aux->LPeriodo), periodo, elementos))
                return 1;
            else
                return 0;

        }
        else if ((aux2 = (struct TTransiente * ) malloc (sizeof(struct TTransiente))) == NULL)
                return 0;
        else {

            aux2->trans = transiente;
            aux2->contTrans = 1;
            aux2->prox = aux;
            if (anterior != NULL)
                anterior->prox = aux2;
            else
                (*l) = aux2;
            aux2->LPeriodo = NULL;
            if (AcrescentarP(&(aux2->LPeriodo), periodo, elementos))
                return 1;
            else
                return 0;
        }

    }
};

/* ========================================================================================== */
int AcrescentarP (TLPeriodo *l, long int periodo, long int elementos)
{
    if (*l == NULL){
    /* === Inserindo no início da lista === */
        if (!(*l = (struct TPeriodo *) malloc(sizeof(struct TPeriodo))))
            return 0;

        (*l)->periodo = periodo;
        if (periodo == elementos){
            (*l)->contDistintos = 1;
            (*l)->contRepetidos = 0;
        }
        else {
            (*l)->contDistintos = 0;
```

```
            (*l)->contRepetidos = 1;
        }
        (*l)->prox = NULL;
        (*l)->LElementos = NULL;
        if (AcrescentarO(&((*l)->LElementos),elementos))
            return 1;
        else
        return 0;

    }
    else {

    /* === Variáveis auxiliares === */
        TLPeriodo aux, aux2,anterior;

        anterior = NULL;
        aux = *l;
        while ((aux != NULL) && (aux->periodo < periodo)) {
            anterior = aux;
            aux = aux->prox;
        }
        if ((aux != NULL) && (aux->periodo == periodo)){

            /* === Periodo  encontrado, adicionando 1 ao contador === */
            if (periodo == elementos)
                aux->contDistintos++;
            else
                aux->contRepetidos++;

            if (AcrescentarO(&(aux->LElementos), elementos))
                return 1;
            else
                return 0;

        }
        else if ((aux2 = (struct TPeriodo * ) malloc (sizeof(struct TPeriodo))) == NULL)
                return 0;
        else {

            aux2->periodo = periodo;
            if (periodo == elementos){
                aux2->contDistintos = 1;
                aux2->contRepetidos = 0;
            }
            else {
                aux2->contDistintos = 0;
                aux2->contRepetidos = 1;
            }
            aux2->prox = aux;
            if (anterior != NULL)
                anterior->prox = aux2;
            else
                (*l) = aux2;
            aux2->LElementos = NULL;
```

```
            if (AcrescentarO(&(aux2->LElementos), elementos))
                return 1;
            else
                return 0;
        }

    }
};

/* ========================================================================================== */
int AcrescentarO (TLElementos *l, long int elementos)
{
    if ((*l) == NULL){
    /* === Inserindo no início da lista === */
        if (!((*l) = (struct TElementos *) malloc(sizeof(struct TElementos))))
            return 0;

        (*l)->elementos = elementos;
        (*l)->contElementos = 1;
        (*l)->prox = NULL;
        return 1;
    }
    else {
    /* === Variáveis auxiliares === */
        TLElementos aux,aux2, anterior;

        anterior = NULL;
        aux = *l;
        while ((aux != NULL) && (aux->elementos < elementos)) {
            anterior = aux;
            aux = aux->prox;
        }
        if ((aux != NULL) && (aux->elementos == elementos)){
            /* === Elemento encontrado, incrementando contador === */
            aux->contElementos++;
            return 1;
        }
        else if ((aux2 = (struct TElementos * ) malloc (sizeof(struct TElementos))) == NULL)
                return 0;
        else {

            aux2->elementos = elementos;
            aux2->contElementos = 1;
            aux2->prox = aux;
            if (anterior != NULL)
                anterior->prox = aux2;
            else
                (*l) = aux2;
            return 1;

        }
    }
};
```

```
/* ================================================================================== */
long int QuantTransPeriodo(TLTransiente lt , long int transiente, long int periodo)
{
    while ((lt!= NULL) && (lt->trans < transiente))
        lt= lt->prox;

    /* === Verificando se encontrou o transiente === */
    if ((lt != NULL) && (lt->trans = transiente)){
        TLPeriodo lp;
        lp = lt->LPeriodo;

        while ((lp != NULL) && (lp->periodo < periodo))
            lp = lp->prox;

        /* === Verificando se encontrou o período === */
        if ((lp != NULL) && (lp->periodo == periodo))
            return (lp->contDistintos + lp->contRepetidos);
        else
            return -1;
    }
    else
        return -1;

}

/* ================================================================================== */
int UnificarP(TLTransiente lOriginal, TLPeriodo *noval)
{
    int i;
    TLElementos le;
    TLPeriodo lp;
    while (lOriginal != NULL){
        lp = lOriginal->LPeriodo;
        while (lp != NULL){
            le = lp->LElementos;
            while (le != NULL){
                for(i=1; i <= le->contElementos; i++)
                    if (!(AcrescentarP(&(*noval), lp->periodo, le->elementos)))
                        return 0;
                le = le->prox;
            }
            lp = lp->prox;
        }
        lOriginal = lOriginal->prox;
    }
    return 1;
}

/* ================================================================================== */
void DestruirLT(TLTransiente *l)
{
    TLTransiente ltaux, ltanterior;

    if ((*l) != NULL) {
```

```
        ltanterior = NULL;
        ltaux = (*l);
        while (ltaux != NULL) {
            ltanterior = ltaux;
            ltaux = ltaux->prox;
            DestruirLP(&ltanterior->LPeriodo);
            free(ltanterior);
        }
        *l = NULL;
    }

};

/* ========================================================================================== */
void DestruirLP(TLPeriodo *l)
{
    TLPeriodo lpaux, lpanterior;

    if ((*l) != NULL) {
        lpanterior = NULL;
        lpaux = (*l);
        while (lpaux != NULL) {
            lpanterior = lpaux;
            lpaux = lpaux->prox;
            DestruirLE(&lpanterior->LElementos);
            free(lpanterior);
        }
        *l = NULL;
    }

};

/* ========================================================================================== */
void DestruirLE(TLElementos *l)
{
   TLElementos leaux, leanterior;

    if ((*l) != NULL) {
        leanterior = NULL;
        leaux = (*l);
        while (leaux != NULL) {
            leanterior = leaux;
            leaux = leaux->prox;
            free(leanterior);
        }
        *l = NULL;
    }

};
```

APÊNDICE B – Turista JVisual

O Turista JVisual é um aplicativo desenvolvido em Java, cujo objetivo é apresentar visualmente as trajetórias realizadas pelo caminhante. Uma vez que foi desenvolvido na linguagem de programação Java, este aplicativo é portável para diferentes arquiteturas de *hardware* e sistemas operacionais.

Ao desenvolvê-lo, mantivemos a execução das caminhadas pelo programa de simulação, criado em C, uma vez que essa linguagem apresenta melhor desempenho que aquela utilizada no desenvolvimento do visualizador. Desta forma, também é possível manter as melhorias já concretizadas no programa de simulação, em vários projetos de mestrado e doutorado.

B.0.4 Interface com o usuário

Na Fig. B.1 podemos observar a *interface* de interação com o usuário.

Há nesse aplicativo três formas de exibição das caminhadas. Na configuração *pausada*, ao clicar no botão *executar*, uma caminhada é desenhada por completo, no entanto, para que a próxima caminhada seja desenhada, faz-se necessário um novo click no botão de execução. Assim, permitimos que o usuário observe cada caminhada por um intervalo de tempo indeterminado.

Na modalidade *continuada*, as caminhadas são desenhadas seqüencialmente com um intervalo de tempo entre elas. Esse intervalo é determinado pelo campo *Tempo entre Execuções*, e pode ser alterado pelo usuário.

Na terceira e última modalidade, denominada *interativa*, ao clicar no botão *executar*, um passo da caminhada é desenhado, assim, podemos visualizar cada deslocamento realizado pelo caminhante. Ao clicar no botão de execução, se a caminhada atual tiver chegado ao fim, uma nova caminhada será iniciada.

Para que as caminhadas sejam executadas, inicialmente é necessário importar o arquivo de caminhadas, que deve ser gerado pelo programa de simulação. Desta forma, podemos concluir

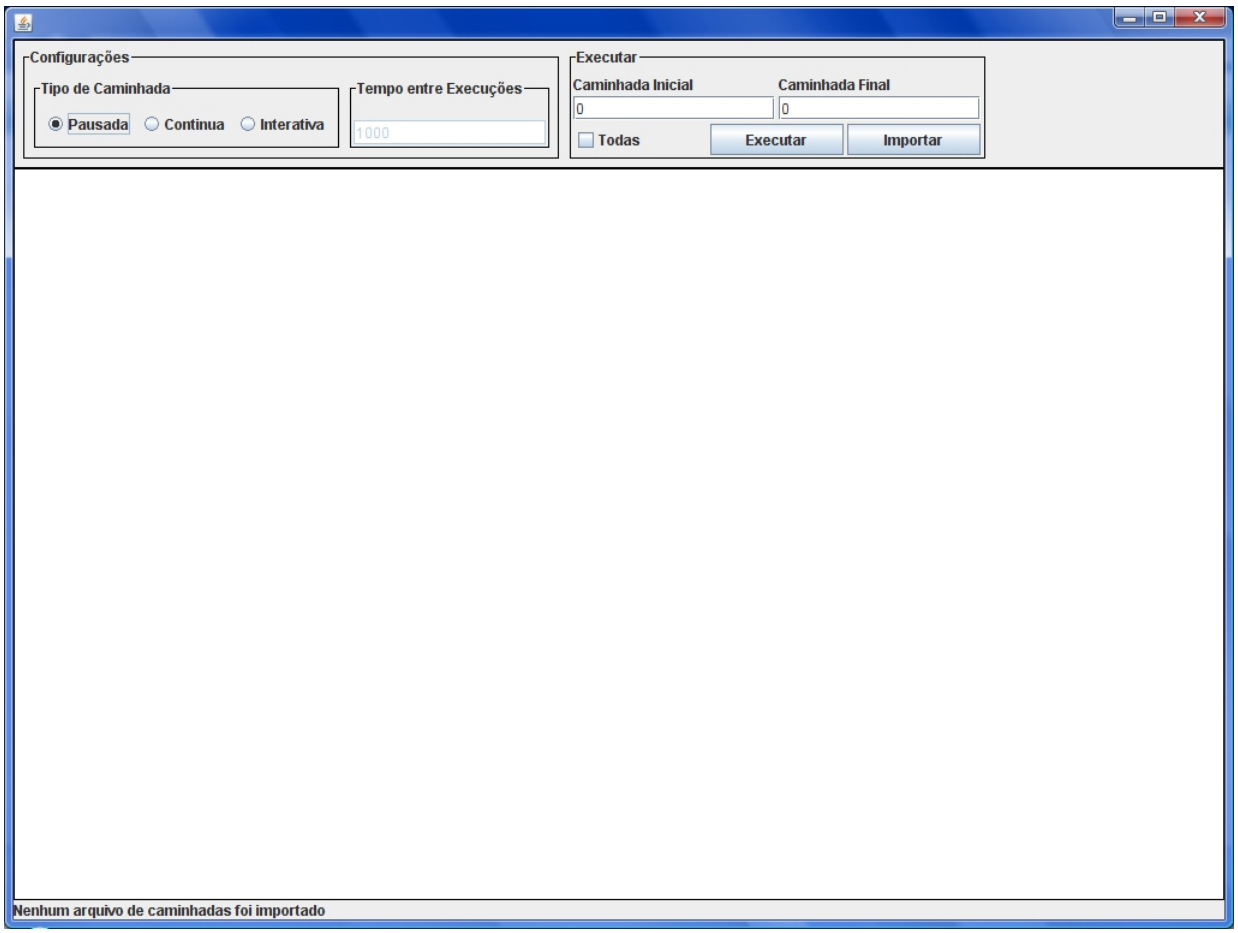


Figura B.1: Tela principal do aplicativo Turista JVisual.

que a integração entre o programa de simulação e o Turista JVisual é realizada através deste arquivo.

Na Fig. B.2 podemos visualizar um trecho de um arquivo de caminhadas obtido ao realizarmos uma simulação no espaço tridimensional.

Nesse arquivo cada caminhada é descrita por um cabeçalho, seguido de um lista de coordenadas cartesianas. O cabeçalho descreve o tempo de transiente e de período de atrator, sendo identificados pelas letras “t” e “p”, respectivamente. O cabeçalho deverá sempre iniciar pelo símbolo “#”.

Também é possível inserir comentários no arquivo de caminhadas, para isso a linha deve iniciar com o caracter “#”. Sendo assim, cabe ao *software* diferenciar entre cabeçalho e comentário.

A importação desse arquivo, pelo Turista JVisual, é realizada através do botão *importar*, que apresentará uma tela de abertura de arquivos, compatível com o sistema operacional utilizado.

Finalmente, é possível também escolhermos quais as caminhadas devem ser desenhadas,

```
#t: 0 p: 3
0.242586        0.145007        0.138744
0.278643        0.155242        0.156502
0.216645        0.155366        0.238254
#=============
#t: 2 p: 3
0.873303        0.600536        0.212086
0.819458        0.629930        0.229740
0.791783        0.495516        0.129227
0.716035        0.406802        0.122751
0.776910        0.532990        0.956465
#=============
#t: 10 p: 3
0.531353        0.444202        0.711130
0.653384        0.428901        0.543688
0.632317        0.359996        0.449970
0.726150        0.396349        0.429613
0.754587        0.335892        0.345272
0.762791        0.229209        0.316060
0.639684        0.176194        0.295057
0.635815        0.138643        0.172547
0.611774        0.086341        0.127955
0.457078        0.117041        0.114125
0.278643        0.155242        0.156502
0.242586        0.145007        0.138744
0.223661        0.068549        0.101247
#=============
#t: 10 p: 3
0.965628        0.316143        0.407296
0.057559        0.393078        0.462940
0.974270        0.548551        0.492000
0.043230        0.574679        0.621862
0.079773        0.748349        0.497223
0.198183        0.866403        0.640706
0.145400        0.735215        0.761115
0.992257        0.863915        0.812534
```

Figura B.2: Arquivo de caminhadas obtido no espaço tridimensional.

para isso, basta utilizarmos os campos *Caminhada Inicial* e *CaminhadaFinal*, que indicará o início e o fim do intervalo de caminhadas que serão executadas. Caso o usuário deseje que todas sejam plotadas, poderá assinalar o campo *todas*.

B.1 Implementação do Turista JVisual

Na Fig. B.3 apresentamos o diagrama de classes de define os aspectos estruturais do programa desenvolvido.

A partir dessa figura, percebemos que a classe principal *VisualTurista* é composta por uma classe abstrata *Visualizador*. A classe *VisualTurista* é responsável por definir o ambiente de desenho, além de coordenar a interação com o usuário e a área de desenho. Já a classe *Visualizador* é responsável por definir a área de desenho e importar as caminhadas, assim, ela deve ser especializada na classes *Visualizador_1D, Visualizador_2D* e *Visualizador_3D*, que implementam a área de desenho para $d = 1, d = 2$ e $d = 3$ respectivamente.

Finalmente, a classe *Caminhada* é responsável por armazenar as caminhadas importados do arquivo de caminhadas descrito na Seção B.2.

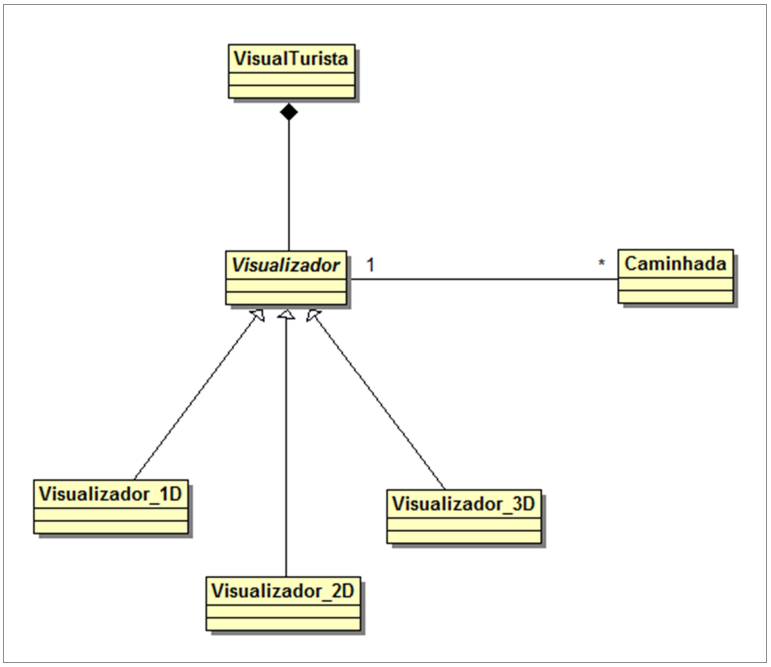


Figura B.3: Diagrama de classes do aplicativo Turista JVisual.

B.1.1 Classe VisualTurista

A seguir é apresentado o código da classe *VisualTurista*, no entanto, retiramos a seção de importação de bibliotecas, a fim de reduzirmos o código apresentado e também pelo fato dessa tarefa ser totalmente automatizada nos ambientes de desenvolvimento.

```
public class VisualTurista extends JFrame{
        private enum TiposCaminhada {PAUSADA, CONTINUA, INTERATIVA};

        private Visualizador visual;
        private JRadioButton setExecPausada;
        private JRadioButton setExecContinua;
        private JRadioButton setExecInterativa;
        private ButtonGroup grupoTipoCaminhada;
        private JTextField campoTempoEspera;
        private JLabel lblCaminhadaInicial;
        private JLabel lblcaminhadaFinal;
        private JTextField campoCaminhadaInicial;
        private JTextField campoCaminhadaFinal;
        private JCheckBox setExecTodasCaminhadas;
        private JButton btnImportarCaminhadas;;
        private JButton btnExecutar;
        private JLabel barraStatus;
        private boolean dadosImportados;

        private TiposCaminhada tipoCaminhada;
        private int indiceCaminhadaPausada; // Utilizado para controlar a execução pausada
    private File nomeArquivo;
```

```
/* Variáveis que controlam o modo interativo */
int int_caminhadaAtual;
int int_quantArcosDesenhar;

/* Construtor da classe */
public VisualTurista(int largura, int altura)
{
        setLayout( new BorderLayout());
        setSize( largura, altura );
        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        /* Criando painel superior */
        JPanel painelSup = new JPanel();
        painelSup.setLayout(new BorderLayout());
        //Border bordaPainelSup = ;
        painelSup.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        painelSup.setVisible(true);

        /* Criando área de desenho */
        visual = new Visualizador_1D( largura, altura,  100);
        visual.setRaioSitio( 5.0 );
        visual.setCorFundo(Color.WHITE);
        visual.setCorTransiente(Color.BLACK);
        visual.setCorPontoInicial(Color.MAGENTA);
        visual.setCorPeriodo(Color.BLUE);
        visual.setCorArcos(Color.BLACK);
        visual.setAlturaInicialArco(50.0);
        visual.setFCAlturaArco(2.0);

        /* Adicionando visualizador ao painel superior */
        painelSup.add(visual, BorderLayout.CENTER);

        /* Criando painel inferior */
        JPanel painelInf = new JPanel();
        FlowLayout layoutPainelInf = new FlowLayout();
        layoutPainelInf.setAlignment(FlowLayout.LEFT);
        painelInf.setLayout(layoutPainelInf);
        Border bordapainelInf = BorderFactory.createLineBorder(Color.BLACK);
        painelInf.setBorder(bordapainelInf);
        painelInf.setVisible(true);

        /* Criando configurações visuais genéricas que será utilizada nos painéis internos */
        Border bordaGenerica = BorderFactory.createLineBorder(Color.BLACK);
        int alturaPadraoPaineisInternos = 60;
        int alturaPadraopaineisExternos = 95;

        /* Criando painel de configuração */
        JPanel painelConfiguracao = new JPanel();
        painelConfiguracao.setLayout(new FlowLayout());
        painelConfiguracao.setBorder(BorderFactory.
           createTitledBorder(bordaGenerica, "Configurações"));
        painelConfiguracao.setPreferredSize(new Dimension(450, alturaPadraopaineisExternos));
```

```
/* Criando Componentes para seleção do tipo de execução */
setExecPausada = new JRadioButton("Pausada");
tipoCaminhada = TiposCaminhada.PAUSADA;
setExecContinua = new JRadioButton("Continua");
setExecInterativa = new JRadioButton("Interativa");
grupoTipoCaminhada = new ButtonGroup();

grupoTipoCaminhada.add(setExecPausada);
grupoTipoCaminhada.add(setExecContinua);
grupoTipoCaminhada.add(setExecInterativa);

/* Registro de eventos para JRadioButtons */
setExecPausada.addItemListener(new RadioButtonHandler());
setExecContinua.addItemListener(new RadioButtonHandler());
setExecInterativa.addItemListener(new RadioButtonHandler());


JPanel painelTipoCaminhada = new JPanel();
painelTipoCaminhada.setBorder(BorderFactory.
   createTitledBorder(bordaGenerica, "Tipo de Caminhada"));
painelTipoCaminhada.add(setExecPausada);
painelTipoCaminhada.add(setExecContinua);
painelTipoCaminhada.add(setExecInterativa);

/* Criando componentes para setar o tempo entre execuções */
campoTempoEspera = new JTextField("1000");
JPanel painelTempoEspera = new JPanel();
painelTempoEspera.setLayout(new BorderLayout());
painelTempoEspera.add(campoTempoEspera, BorderLayout.SOUTH);
painelTempoEspera.setBorder(BorderFactory.
    createTitledBorder(bordaGenerica, "Tempo entre Execuções"));
Dimension tamanhoPainelTempoEspera = new Dimension(170,alturaPadraoPaineisInternos);
painelTempoEspera.setPreferredSize(tamanhoPainelTempoEspera);

/* Adicionando componentes ao painel de configuração */
painelConfiguracao.add(painelTipoCaminhada);
painelConfiguracao.add(painelTempoEspera);

/* Criando painel de Execução.
 * Este painel é dividido em dois outros painéis */
JPanel painelExecutar = new JPanel();
painelExecutar.setBorder(BorderFactory.createTitledBorder(bordaGenerica, "Executar"));
painelExecutar.setPreferredSize(new Dimension(350, alturaPadraopaineisExternos));
painelExecutar.setLayout(new BorderLayout());

/* Criando componentes do painel de execução */
lblCaminhadaInicial = new JLabel("Caminhada Inicial");
lblcaminhadaFinal = new JLabel("Caminhada Final");
campoCaminhadaInicial = new JTextField("0");
campoCaminhadaFinal = new JTextField("0");
setExecTodasCaminhadas = new JCheckBox("Todas");

/* Vinculando evento para JTextField */
TextFieldHandler handlerTextFiel = new TextFieldHandler();
```

```
        campoCaminhadaInicial.addFocusListener(handlerTextFiel);

        /* Vinculando evento para JCheckBox */
        setExecTodasCaminhadas.addItemListener(new CheckBoxHandler());

        btnImportarCaminhadas = new JButton("Importar");
        btnExecutar = new JButton("Executar");

        /* Registrando handler de eventos para JButtons */
        ButtonHandler handlerButton= new ButtonHandler();
        btnImportarCaminhadas.addActionListener(handlerButton);
        btnExecutar.addActionListener(handlerButton);

        /* Criando painel de execução superior e adicionando seus componentes */
        JPanel painelExecSup = new JPanel();
        painelExecSup.setLayout(new GridLayout(0,2,3,0));
        painelExecSup.add(lblCaminhadaInicial);
        painelExecSup.add(lblcaminhadaFinal);
        painelExecSup.add(campoCaminhadaInicial);
        painelExecSup.add(campoCaminhadaFinal);

        /* Criando painel de execução inferior e adicionando seus componentes */
        JPanel painelExecInf = new JPanel();
        painelExecInf.setLayout(new GridLayout(0,3,3,0));
        painelExecInf.add(setExecTodasCaminhadas);
        painelExecInf.add(btnExecutar);
        painelExecInf.add(btnImportarCaminhadas);

        /* Adicionando paineis superior e inferior ao painel de execução */
        painelExecutar.add(painelExecSup, BorderLayout.NORTH);
        painelExecutar.add(painelExecInf, BorderLayout.SOUTH);

        /* Adicionando paineis internos ao painel inferior da tela */
        painelInf.add(painelConfiguracao);
        painelInf.add(painelExecutar);

        /* Criando barra de status */
        barraStatus = new JLabel("Nenhum arquivo de caminhadas foi importado");

        /* Inserindo paineis e a barra de status na janela principal */
        add(painelInf, BorderLayout.NORTH);
        add(painelSup, BorderLayout.CENTER);
        add(barraStatus, BorderLayout.SOUTH);

        /* Configurando o tipo de caminhada inicialmente selecionado */
        setExecPausada.setSelected(true);
        dadosImportados = false;
}

private void int_reiniciarQuantidadeArcos()
{
        int_quantArcosDesenhar = 0;
}
```

```
/* Classe interna para tratar eventos de JRadioButtons */
private class RadioButtonHandler implements ItemListener
{
        public void itemStateChanged(ItemEvent event)
        {
                if(event.getSource() == setExecPausada){
                        tipoCaminhada = TiposCaminhada.PAUSADA;
                        campoTempoEspera.setEnabled(false);
                        indiceCaminhadaPausada = Integer.parseInt(campoCaminhadaInicial.getText());
                }

                if(event.getSource() == setExecContinua){
                        tipoCaminhada = TiposCaminhada.CONTINUA;
                        campoTempoEspera.setEnabled(true);
                }

                if(event.getSource() == setExecInterativa){
                        tipoCaminhada = TiposCaminhada.INTERATIVA;
                        campoTempoEspera.setEnabled(false);
                        int_caminhadaAtual = Integer.parseInt(campoCaminhadaInicial.getText());
                        int_reiniciarQuantidadeArcos();

                }
        }
} // Fim classe RadioButtonHandler

/* Classe que implementa o Handler para JTextField */
private class TextFieldHandler implements FocusListener
{
        public void focusGained(FocusEvent event)
        {

        }

        public void focusLost(FocusEvent event)
        {
                if(event.getSource() == campoCaminhadaInicial){
                        if(setExecInterativa.isSelected()){
                                int_caminhadaAtual = Integer.parseInt(campoCaminhadaInicial.getText());
                                int_reiniciarQuantidadeArcos();
                        }
                }
        }
}

/* Classe que implementa o Handler para JCheckBox */
private class CheckBoxHandler implements ItemListener
{
        public void itemStateChanged(ItemEvent event)
        {
                if (event.getSource() == setExecTodasCaminhadas){
                        if(dadosImportados){
                                if(setExecTodasCaminhadas.isSelected()){
```

```
                                        campoCaminhadaInicial.setEnabled(false);
                                        campoCaminhadaFinal.setEnabled(false);
                                        campoCaminhadaInicial.setText("0");
                                        campoCaminhadaFinal.setText(String.
                                            valueOf(visual.quantidadeCaminhadas() - 1));

                                }
                                else{
                                        campoCaminhadaInicial.setText("0");
                                        campoCaminhadaFinal.setText("0");
                                        campoCaminhadaInicial.setEnabled(true);
                                        campoCaminhadaFinal.setEnabled(true);

                                }

                                if(setExecInterativa.isSelected()){
                                        int_caminhadaAtual = Integer.
                                            parseInt(campoCaminhadaInicial.getText());
                                        int_reiniciarQuantidadeArcos();
                                }
                        }
                        else{
                                JOptionPane.showMessageDialog(null,
                                    "É necessário importar as caminhadas",
                                    "Mensagem", JOptionPane.WARNING_MESSAGE);
                                setExecTodasCaminhadas.setSelected(false);
                        }


                }
        }
} // Fim classe CheckBoxHandler

private class ButtonHandler implements ActionListener
{
        public void actionPerformed(ActionEvent event)
        {

                /* Botão ImportarCaminhadas pressionado */
                if(event.getSource() == btnImportarCaminhadas){
                        JFileChooser dialogoAbrir = new JFileChooser();

                        dialogoAbrir.setFileSelectionMode(JFileChooser.FILES_ONLY);
                        int retorno = dialogoAbrir.showOpenDialog(null);

                        if(retorno == JFileChooser.APPROVE_OPTION){
                                nomeArquivo = dialogoAbrir.getSelectedFile();

                                /* Validando o nome do arquivo */
                                if((nomeArquivo == null) || (nomeArquivo.getName().equals("")))
                                        JOptionPane.showMessageDialog(null,
                                            "O nome do arquivo é inválido",
                                            "Mensagem", JOptionPane.WARNING_MESSAGE);
                                else if(visual.importarDados(nomeArquivo.toString())){
```

```
                        dadosImportados =  true;
                        barraStatus.setText("Arquivo Importado: " +
                            nomeArquivo.toString());
                    }
                    else
                        JOptionPane.showMessageDialog(null, "Erro ao abrir o arquivo
                             de caminhadas", "Informação", JOptionPane.ERROR_MESSAGE);
            }
    }

    /* Botão Executar pressionado */
    if(event.getSource() == btnExecutar){
            if(dadosImportados){

                    /* Execução continuada selecionada */
                    if(setExecContinua.isSelected()){
                            for(int i=Integer.parseInt(campoCaminhadaInicial.getText());
                                i <=Integer.parseInt(campoCaminhadaFinal.getText());
                                i++){
                                    visual.executarCaminhada(i);

                                    try {
                                            Thread.sleep(Long.
                                                parseLong(campoTempoEspera.getText()));
                                    } catch (InterruptedException e) {
                                            JOptionPane.showMessageDialog(null,
                                                "Erro ao paralizar por tempo determinado",
                                                "Erro", JOptionPane.ERROR_MESSAGE);
                                            break;
                                    }
                            }
                    }


                    /* Execução pausada Selecionada */
                    if(setExecPausada.isSelected()){
                            visual.executarCaminhada(indiceCaminhadaPausada);
                            indiceCaminhadaPausada =((indiceCaminhadaPausada + 1) %
                            (Integer.parseInt(campoCaminhadaFinal.getText()) + 1)) +
                                Integer.parseInt(campoCaminhadaInicial.getText());
                    }

                    if(setExecInterativa.isSelected()){
                            visual.executarCaminhada(int_caminhadaAtual,
                                int_quantArcosDesenhar);
                            int_quantArcosDesenhar++;

                            if(int_quantArcosDesenhar > visual.listaCaminhadas.
                                get(int_caminhadaAtual).QuantidadeCoordenadas()){
                                    int_caminhadaAtual = ((int_caminhadaAtual + 1) %
                                        (Integer.parseInt(campoCaminhadaFinal.
                                            getText()) + 1)) +
                                         Integer.parseInt(campoCaminhadaInicial.
                                             getText());
```

```
                                                int_quantArcosDesenhar = 0;
                                        }
                                }
                        }
                        else
                                JOptionPane.showMessageDialog(null, "É necessário importar
                                     as caminhadas", "Mensagem",
                                     JOptionPane.WARNING_MESSAGE);
                }



        } // Fim método actionPerformed
    } // Fim classe ButtonHandler


    public static void main(String args[]) throws InterruptedException
    {
        VisualTurista app = new VisualTurista(1024, 768);
        app.setVisible(true);
    }
}
```

B.1.2 Classe abstrata Visualizador

A seguir apresentamos o código da classe abstrata *Visualizador*.

```
public abstract class Visualizador extends JPanel
{
    /* Declaracao de constantes */
    protected enum Direcoes {DIREITA, ESQUERDA};


    /* Variáveis de configuração do ambiente de desenho */
    private Color corFundo = Color.WHITE;
    private Color corTransiente;
    private Color corPontoInicial;
    private Color corPeriodo;
    private Color corArcos;
    private double raioSitio;
    private int borda;
    private double alturaInicialArco; // Altura inicial do arco
    private double fcAlturaArco;           // Fator de ajuste da altura do arco
    private boolean desenhar;


    /* Variáveis de suporte à caminhada */

    /** Lista das caminhadas armazenadas no arquivo */
    protected ArrayList<Caminhada> listaCaminhadas;
```

```
/* Variáveis que controlam o método paint() */
private int caminhadaInicial;
private int caminhadaFinal;
private int indiceCaminhada;

// Controla a quantidade de arcos desenhados no método intetativo.
private int quantidadeArcosDesenhar;

private boolean dadosImportados;

/* Construtor da super classe */
public Visualizador(int largura, int altura, int borda)
{
        /* configurando área de desenho */
        setSize(largura, altura);
        setBorda(borda);

        /* Inicializando os atributos */
        caminhadaFinal = caminhadaInicial = indiceCaminhada = 0;
        //modoInterativo = false;
        dadosImportados = false;
        desenhar = false;
        alturaInicialArco = getWidth() * 0.10;
        fcAlturaArco = 5.0;
        raioSitio = 2.0;
        corFundo = Color.WHITE;
        corTransiente = Color.BLACK;
        corPontoInicial = Color.ORANGE;
        corPeriodo = Color.MAGENTA;
        corArcos = Color.BLACK;
}


/* Métodos concretos: Setters e Getters */
public void setBorda(int borda)
{
        this.borda = borda;
}

public int getBorda()
{
        return borda;
}

public Color getCorFundo()
{
        return corFundo;
}

public void setCorFundo(Color cor)
{
        corFundo = cor;
}
```

```
public Color getCorTransiente()
{
        return corTransiente;
}
public void setCorTransiente(Color cor)
{
        corTransiente = cor;
}

public Color getCorPontoInicial()
{
        return corPontoInicial;
}

public void setCorPontoInicial(Color cor)
{
        corPontoInicial = cor;
}

public Color getCorPeriodo()
{
        return corPeriodo;
}

public void setCorPeriodo(Color cor)
{
        corPeriodo = cor;
}

public Color getCorArcos()
{
        return corArcos;
}

public void setCorArcos(Color cor)
{
        corArcos = cor;
}

public double getAlturaInicialArco()
{
        return alturaInicialArco;
}

public void setAlturaInicialArco(double altura)
{
        alturaInicialArco = altura;
}

public double getFCAlturaArco()
{
        return fcAlturaArco;
}
```

```
public void setFCAlturaArco(double altura)
{
        fcAlturaArco = altura;
}

public void setRaioSitio(double raio)
{
        raioSitio = raio;
}

public double getRaioSitio()
{
        return raioSitio;
}

public int getCaminhadaInicial()
{
        return caminhadaInicial;
}

public int getCaminhadaFinal()
{
        return caminhadaFinal;
}

public int getindiceCaminhada()
{
        return indiceCaminhada;
}

public int getQuantidadeArcosDesenhar()
{
        return quantidadeArcosDesenhar;
}

protected void setDadosImportados(boolean valor)
{
        dadosImportados = valor;
}

protected boolean getDadosImportados()
{
        return dadosImportados;
}

public boolean getDesenhar()
{
        return desenhar;
}

public void setDesenhar(boolean desenhar)
{
        this.desenhar = desenhar;
```

```
        }

        /* Métodos concretos: Manipulação da caminhada */
        public void InicilizarCaminhada()
        {
                listaCaminhadas = new ArrayList< Caminhada >();
        }

        public int quantidadeCaminhadas()
        {
                return listaCaminhadas.size();
        }

        public void executarCaminhada(int indice)
        {
                if((indice >= 0) && (indice < listaCaminhadas.size())){
                        indiceCaminhada = indice;
                        quantidadeArcosDesenhar = listaCaminhadas.get(indiceCaminhada).
                            QuantidadeCoordenadas();
                        paintImmediately(0, 0, getWidth(), getHeight());
                }
                else
                        JOptionPane.showMessageDialog(null, "Índice da caminhada inválido",
                            "Mensagem", JOptionPane.ERROR_MESSAGE);
        }


        public void executarCaminhada( int indice, int quantidadeArcos )
        {
                if((indice >= 0) && (indice < listaCaminhadas.size())){
                        indiceCaminhada = indice;

                        if((quantidadeArcos >= 0) && (quantidadeArcos <=
                            listaCaminhadas.get(indiceCaminhada).QuantidadeCoordenadas())){
                                quantidadeArcosDesenhar = quantidadeArcos;
                                paintImmediately(0, 0, getWidth(), getHeight());
                        }
                        else
                                JOptionPane.showMessageDialog(null, "Quantidade de arcos inválido",
                                    "Mensagem", JOptionPane.ERROR_MESSAGE);
                }
                else
                        JOptionPane.showMessageDialog(null, "Índice da caminhada inválido",
                            "Mensagem", JOptionPane.ERROR_MESSAGE);

        }

    /* Métodos abstratos.
         *
         * Os métodos aqui listados dependem da dimensão
         * para serem corretamente implementados. */
        protected abstract boolean importarDados(String endEntrada);
}
```

B.1.3 Classe Visualizador_1D

Para fins deste trabalho, necessitamos apenas da implementação da classe *Visualizador_1D*, que apresenta-se como uma das mais complexas, visto que a área de desenho é um plano bidimensional. Desta forma, deixamos para trabalhos futuros as implementações das classes *Visualizador_2D* e *Visualizador_3D*.

```
public class Visualizador_1D extends Visualizador
{

  /* Construtor */
        public Visualizador_1D( int largura, int altura, int borda )
        {
                super( largura, altura, borda );
        }


        /* Método responsável por exibir a caminhada */
        public void paint( Graphics g )
        {


                Graphics2D g2 = ( Graphics2D ) g;


    Caminhada caminhadaAtual;
                Double[] coordExtremas = new Double[2];
                Double[] pontoAtual;
                Ellipse2D desenhoPonto;
                Direcoes direcao;


                /* Definindo a área de desenho */
                g2.setColor( getCorFundo() );
                g2.fillRect( 0, 0, getWidth(), getHeight());


                /* Realiza-se o desenho somente se há dados importados de um arquivo */
                if( getDadosImportados() ){
                        caminhadaAtual = listaCaminhadas.get(getindiceCaminhada()).cloneCaminhada();
                        coordExtremas = coordenadasExtremas(caminhadaAtual);
                        if( coordExtremas != null ){

                                /* Inserindo o ponto que finalizará o período.
                                 * O ponto a ser inserido é aquele armazendado na posicao trans,
                                 * uma vez que este é o primeiro ponto do período, já que os vetores
                                 * em Java iniciam na posição zero.
                                 * É importante lembrar que esse ponto deve ser inserido apenas uma vez,
                                 * já que o método paint() é executado várias vezes*/
                                pontoAtual = new Double[1];
                                pontoAtual[0] = caminhadaAtual.getPonto(caminhadaAtual.getTrans())[0];
```

```
caminhadaAtual.inserirPonto(pontoAtual);

/* Dimensionando os pontos para que reflitam o tamanho da área de desenho,
 * lembrando que há um ponto a mais no vetor caminhadaAtual que indica o
 * fechamento da caminhada */
for(int j = 0; j < caminhadaAtual.QuantidadeCoordenadas(); j++){
        pontoAtual =  DimensionarPonto(caminhadaAtual.getPonto(j),
            coordExtremas, getWidth(), getBorda());

        caminhadaAtual.setCoordenada(j, pontoAtual);
}

/* Desenhando o ponto inicial */
g2.setColor(getCorPontoInicial());
desenhoPonto = new Ellipse2D.Double( caminhadaAtual.getPonto(0)[0] -
    getRaioSitio(), (getHeight()/2) - getRaioSitio(),
    2 * getRaioSitio(), 2 * getRaioSitio() );

g2.draw(desenhoPonto);

/* Desenhando coordenadas pontos do transiente */
g2.setColor( getCorTransiente());
for(int j = 1; j < caminhadaAtual.getTrans(); j++){
  desenhoPonto = new Ellipse2D.Double( caminhadaAtual.getPonto(j)[0] -
      getRaioSitio(), (getHeight()/2) - getRaioSitio(),
      2 * getRaioSitio(), 2 * getRaioSitio() );

        g2.draw(desenhoPonto);
}

/* Desenhando pontos do período */
g2.setColor(getCorPeriodo());
for(int j = caminhadaAtual.getTrans();
    j < caminhadaAtual.QuantidadeCoordenadas(); j++){
        if(caminhadaAtual.getPonto(j)[0] != caminhadaAtual.
          getPonto(0)[0]){

                desenhoPonto = new Ellipse2D.Double(caminhadaAtual.
                  getPonto(j)[0] - getRaioSitio(), (getHeight()/2) -
                  getRaioSitio(), 2*getRaioSitio(),2*getRaioSitio());

                g2.draw(desenhoPonto);
        }
}


/* Desenhando os arcos
 *
 * Faz-se o desenho apenas se houver dois ou mais pontos.
 * Vale lembrar também que caso o modo interativo esteja lidado,
 * somente alguns arcos serão desenhados */
QuadCurve2D arcoAtual;
double coordxInicial, coordxFinal, coordyArco;
boolean mudouDirecao = false;
```

```
double alturaArco = getAlturaInicialArco();

//g2.setColor(getCorArcos());
if(caminhadaAtual.QuantidadeCoordenadas() >= 2){
        if(caminhadaAtual.getPonto(0)[0] < caminhadaAtual.getPonto(1)[0])
                direcao = Direcoes.DIREITA;
        else
                direcao = Direcoes.ESQUERDA;

        for(int j = 1; j <= getQuantidadeArcosDesenhar(); j++){
                /* verificando se houve mudança de direção e a
                 * necessidade de aumentar a amplitude do arco */
                if((caminhadaAtual.getPonto(j-1)[0] < caminhadaAtual.
                  getPonto(j)[0])&& (direcao == Direcoes.ESQUERDA)){
                        mudouDirecao = true;
                        direcao = Direcoes.DIREITA;
                }

                if((caminhadaAtual.getPonto(j-1)[0] >= caminhadaAtual.
                  getPonto(j)[0]) && (direcao == Direcoes.DIREITA)){
                        mudouDirecao = true;
                        direcao = Direcoes.ESQUERDA;
                }

                if((mudouDirecao) && (direcao == Direcoes.DIREITA)){
                        alturaArco *= getFCAlturaArco();
                        mudouDirecao = false;
                }

                coordxInicial = caminhadaAtual.getPonto(j - 1)[0];
                coordxFinal = caminhadaAtual.getPonto(j)[0];
                coordyArco = (caminhadaAtual.getPonto(j-1)[0] <
                  caminhadaAtual.getPonto(j)[0] ?
                  getHeight() /2 - alturaArco        :
                  getHeight() /2 + alturaArco);

                arcoAtual = new QuadCurve2D.Double(coordxInicial,
                  getHeight()/2, (coordxFinal + coordxInicial) /2,
                  coordyArco, coordxFinal, getHeight()/2);

                /* Ajustando a cor do arco a ser desenhado.
                 * Isto é utilizado para separar os arcos dos transientes
                 * daqueles para períodos */
                if(j <= caminhadaAtual.getTrans())
                        g2.setColor(getCorTransiente());
                else
                        g2.setColor(getCorPeriodo());

                g2.draw(arcoAtual);

        }
}
else
        JOptionPane.showMessageDialog(null,
```

```
                                "A caminhada não possui coordenadas",
                                "Informação", JOptionPane.INFORMATION_MESSAGE);
                }
        }
} // Fim método Paint


/* Esta funcão retorna a menor e a maior coordenadas da caminhada.
 * Como retorno é utilizado um tipo Double[], sendo que a posicao 0
 * armazena a menor coordenada, enquanto a posicao 1 armazena a maior */
private Double[] coordenadasExtremas(Caminhada caminhada)
{
        Double[] coordAtual;
        Double[] coordExtremas = new Double[2];

        if( caminhada.QuantidadeCoordenadas() > 0){
                coordExtremas[0] = coordExtremas[1] = caminhada.getPonto( 0 )[0];

                for( int i = 1; i < caminhada.QuantidadeCoordenadas(); i++ ){
                        coordAtual = caminhada.getPonto( i );

                        if( coordAtual[0] < coordExtremas[0] )
                                coordExtremas[0] = coordAtual[0];
                        else if( coordAtual[0] > coordExtremas[1] )
                                coordExtremas[1] = coordAtual[0];
                }
                return coordExtremas;
        }
        else
                return null;
}

private Double[] DimensionarPonto(Double[] ponto, Double[] coordExtremas,
                                  int tamanhoTela, int borda )
{
        ponto[0] = (( tamanhoTela - 2 * borda ) * (( ponto[0] - coordExtremas[0]) /
          (coordExtremas[1] - coordExtremas[0]))) + borda;
        return ponto;
}


/* Método responsável por importar os dados */
protected boolean importarDados(String endEntrada)
{
        Scanner entrada;
        String linha;
        int posicaoInicial, posicaoFinal;
        int transAtual, periodoAtual;

        Caminhada caminhadaAtual = null;
        Double[] pontoAtual;

        InicilizarCaminhada();
        try
```

```
        {
                entrada = new Scanner( new File(endEntrada) );
        }
        catch(FileNotFoundException fileNotFoundException)
        {
                setDadosImportados( false );
                return(false);
        }

        /* Obtendo registros do arquivo */
        while(entrada.hasNext())
        {
                linha = entrada.nextLine();
                if(!linha.isEmpty()){
                        /* Formatando a linha */
                        linha = linha.toLowerCase();
                        linha = linha.replaceAll( " ", "" );
                        linha = linha.replaceAll( "", "" );

                        if(linha.startsWith("#=")){
                                /* Linha que separa as caminhadas, o que indica que
                                 * uma nova caminhada será inicializada e que a
                                 * caminhada anterior deverá ser armanzenada */
                                if(caminhadaAtual != null)
                                        listaCaminhadas.add(caminhadaAtual);
                        }
                        else if(linha.startsWith("#")){
                                // Armazenando o transiente e período
                                posicaoInicial = linha.indexOf("t:");
                                posicaoFinal = linha.indexOf("p:");
                                transAtual = Integer.parseInt( linha.substring(posicaoInicial + 2,
                                  posicaoFinal ) );

                                posicaoInicial = linha.indexOf("p:");
                                posicaoFinal = linha.length();
                                periodoAtual = Integer.parseInt( linha.substring( posicaoInicial + 2,
                                  posicaoFinal ) );

                                //Criando uma nova caminhada
                                caminhadaAtual = new Caminhada(transAtual, periodoAtual);
                        }
                        else{
                                // Armazenando a caminhada
                                posicaoFinal = linha.indexOf("-");
                                pontoAtual = new Double[1];
                                pontoAtual[0] = Double.parseDouble(linha.substring(0,posicaoFinal));
                                caminhadaAtual.inserirPonto(pontoAtual);
                        }
                }
        } //while(entrada.hasNext()

        setDadosImportados( true );
        return true;
}
```

```
} // fim da classe
```

B.1.4 Classe Caminhada

Finalmente, a seguir listamos o código da classe *Caminhada*.

```
public class Caminhada {
        private ArrayList<Double[]> coordenadas;
        private int transiente;
        private int periodo;

        /* Construtor */
        public Caminhada( int transiente, int periodo )
        {
                this.transiente = transiente;
                this.periodo = periodo;

                coordenadas = new ArrayList< Double[] >();

        }

        /* Métodos de manipulação do transiente */
        public void setTrans( int valor )
        {
                transiente = valor;
        }

        public int getTrans()
        {
                return transiente;
        }

         /* Métodos de manipulação do período */
        public void setPeriodo( int valor )
        {
                periodo = valor;
        }

        public int getPeriodo()
        {
                return periodo;
        }


        /* Métodos de manipulação do ArrayList de coordenadas */
        public int QuantidadeCoordenadas()
        {
                return coordenadas.size();
```

```
}

public boolean inserirPonto( Double[] ponto )
{
        return ( coordenadas.add(ponto) );
}


public Double[] getPonto( int posicao )
{
        if ( posicao <= coordenadas.size() )
                return coordenadas.get( posicao );
        else
                return null;
}

public void setCoordenada(int posicao, Double[] valor)
{
        coordenadas.set(posicao, valor);
}

public Caminhada cloneCaminhada()
{
        Caminhada caminhadaClone = new Caminhada(transiente, periodo);
        caminhadaClone.coordenadas = (ArrayList< Double[] >)coordenadas.clone();
        return caminhadaClone;

}
} // Fim da classe Caminhada
```

Referências

[1] J. B. Bastos Filho, "Causalidade, (in)determinismo e (im)previsibilidade. por que o conceito de causa é tão importante?," *Revista Brasileira de Ensino de Física*, vol. 30, junho 2008.

[2] M. Paty, "A noção de determinismo na física e seus limites," *Revista Latino-Americana de Filosofia e História da Ciência*, vol. 2, pp. 465–492, dezembro 2004.

[3] J. Gleick, *Caos: A Criação de uma Nova Ciência*. Rio de Janeiro, RJ: Elsevier, 17 ed., 1989.

[4] C. H. Rycroft and M. Z. Basant, "Introduction to random walks and diffusion," *Lecture Notes, MIT*, february 2005.

[5] M. Z. Bazant, "From random walks to continuum diffusion," *Lecture Notes, MIT*, february 2006.

[6] N. Grazyna and E. Gert, "Asymmetric one-dimensional random walks," *J. Chem. Phys.*, vol. 129, pp. 1–4, 2008.

[7] M. Z. Bazant, "Historical introduction to random walks and diffusion," *Lecture Notes for Spring, MIT*, february 2001.

[8] O. Kinouchi, A. Martinez, G. Lima, G. Lourenço, and S. Risau-Gusman, "Deterministic walks in random networks: an application to thesaurus graphs," *Physica A Statistical Mechanics and its Applications*, vol. 315, pp. 665–676, 2002.

[9] D. Boyer, O. Miramontes, G. Ramos-Fernández, J. L. Mateos, and G. Cocho, "Modeling the searching behavior of social monkeys," *Physica A Statistical Mechanics and its Applications*, vol. 342, pp. 329–335, 2004.

[10] J. P. Boon, "Random and deterministic walks on lattices," 2005.

[11] D. S. Fisher, "Random walks in random environments," *Phys. Rev. A*, vol. 30, pp. 960–964, 1984.

[12] J. D. Noh and H. Rieger, "Constrained spin-dynamics description of random walks on hierarchical scale-free networks," *Phys. Rev. E*, vol. 69, pp. 1–8, 2004.

[13] E. Brunet and B. Derrida, "An exactly soluble noisy traveling wave equation appearing in the problem of directed polymers in a random medium," *Phys. Rev. E*, vol. 70, pp. 1–5, 2004.

[14] D. Boyer and H. Larralde, "Looking for the right thing at the right place: Phase transition in an agent model with heterogeneous spatial resources," *Complexity*, vol. 10, pp. 52–55, 2005.

[15] A. Fronczak and P. Fronczak, "Biased random walks on complex networks: the role of local navigation rules," 2007.

[16] E. Agliari, M. Casartelli, and A. Vezzani, "Interacting random walkers and non-equilibrium fluctuations," *Eur. Phys. J. B.*, vol. 65, pp. 1–8, 2008.

[17] G. F. Lima, A. S. Martinez, and O. Kinouchi, "Deterministic walks in random media," *Phys. Rev. Lett.*, vol. 87, p. 010603, 2001.

[18] A. S. Martinez, O. Kinouchi, and S. Risau-Gusman, "Exploratory behavior, trap models, and glass transitions," *Phys. Rev. E*, vol. 69, no. 1, p. 017101, 2004.

[19] C. A. Sangaletti Terçariol, R. Silva González, Tavares Reis Oliveira, W., and A. Souto Martinez, "Deterministic and random partially self-avoiding walks in random media," *Phys. A.*, vol. 386, pp. 678–680, 2007.

[20] M. C. Santos, D. Boyer, O. Miramontes, G. M. Viswanathan, E. P. Raposo, J. L. Mateos, and M. G. E. da Luz, "Origin of power-law distributions in deterministic walks: The influence of landscape geometry," *Phys. Rev. E.*, vol. 75, p. 061114, 2007.

[21] D. Boyer, "Intricate dynamics of a deterministic walk confined in a strip," 2008.

[22] C. A. S. Tercariol, "Resultados analíticos para as distribuições estatísticas relacionadas à caminhada determinista do turista sem memória: Efeito da dimensionalidade do sistema e modelos de campo médio," Master's thesis, FFCLRP, USP, Ribeirão Preto, SP, novembro 2004.

[23] G. F. Lima, *Caminhadas determinísticas em meios desordenados: problema da caminhada do turista*. PhD thesis, FFCLRP, USP, Ribeirão Preto, SP, junho 2002.

[24] Derrida, B. and Flyvbjerg, H. , "The random map model: a disordered model with deterministic dynamics," *Journal de Physique*, vol. 48, pp. 971–978, 1987.

[25] C. A. S. Tercariol and A. S. Martinez, "Analytical results for the statistical distribution related to a memoryless deterministic walk: Dimensionality effect and mean-field models," *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, vol. 72, no. 2, p. 021103, 2005.

[26] C. A. S. Tercariol, *Caminhadas Deterministas Parcialmente Auto-repulsivas: Resultados Analíticos para o Efeito da Memória do Turista na Exploração de Meios Desordenados*. PhD thesis, FFCLRP, USP, Ribeirão Preto, SP, dezembro 2008.

[27] R. S. G. González, "Difusão anômala: transição entre os regimes localizado e estendido na caminhada do turista unidimensional," Master's thesis, FFCLRP, USP, Ribeirão Preto, SP, agosto 2006.

[28] R. W. Sebesta, *Conceitos de Linguagem de Programação*. Bookman, 5 ed., 2003.

[29] R. Jain, *The Art of Computer Systems Performance Analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley, 1991.

[30] M. M. d. Oliveira and G. A. Braga, "O fenômeno de transição de fase no modelo de percolação de elos* em d dimensões," *Revista Brasileira de Ensino de Física*, vol. 24, pp. 448–454, Dezembro 2002.

[31] M. G. Pereira, G. Corso, L. L.S., and F. J. E., "Percolation properties and universality class of a multifractal random tiling," *International Journal of Modern Physics C*, vol. 16, pp. 317 – 325, February 2005.

[32] B. Berkowitz and R. P. Ewing, "Percolation theory and network modeling applications in soil physics," *Surveys in Geophysics*, vol. 19, pp. 23 – 72, january 1998.

[33] K. M. Golden, S. F. Ackley, and V. Lytle, "The percolation phase transition in sea ice," *Science*, vol. 19, pp. 23 – 72, December 1998.

[34] H. Allen, "A new conceptual model for forest fires based on percolation theory," *Complex.*, vol. 13, no. 3, pp. 12–17, 2008.

[35] M. E. J. Newman, "Spread of epidemic disease on networks," *Phys. Rev. E*, vol. 66, p. 016128, Jul 2002.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo