UNIVERSIDADE ESTADUAL PAULISTA JÚLIO DE MESQUITA FILHO
FACULDADE DE CIÊNCIAS DE BAURU - DEPARTAMENTO DE COMPUTAÇÃO

CAMINHO ÓTIMO EM MALHA DINÂMICA BIDIMENSIONAL

RAFAEL NUNES CASEIRO

Bauru

Janeiro de 2023

RAFAEL NUNES CASEIRO

CAMINHO ÓTIMO EM MALHA DINÂMICA BIDIMENSIONAL

Trabalho de Conclusão de Curso apresentado junto ao Curso de Bacharelado em Ciências da Computação da Universidade Estadual Paulista Júlio de Mesquita Filho - Faculdade de Ciências de Bauru, Departamento de Computação, como requisito parcial à obtenção do título de Bacharel em Ciências da Computação.

Orientadora:

Profª. Drª. Simone das Graças Domingues Prado.

Bauru

Janeiro de 2023

C337c

Caseiro, Rafael Nunes
Caminho Ótimo em Malha Dinâmica Bidimensional / Rafael Nunes Caseiro. -- Bauru, 2023
29 p. : il.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Faculdade de Ciências, Bauru
Orientadora: Simone das Graças Domingues Prado

1. Busca. 2. Caminho Ótimo. 3. Dijkstra. 4. A*. 5. Malha Dinâmica. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências, Bauru. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

RAFAEL NUNES CASEIRO

CAMINHO ÓTIMO EM MALHA DINÂMICA BIDIMENSIONAL

Trabalho de Conclusão de Curso apresentado junto ao Curso de Bacharelado em Ciências da Computação da Universidade Estadual Paulista Júlio de Mesquita Filho - Faculdade de Ciências de Bauru - Departamento de Computação, como requisito parcial à obtenção do título de Bacharel em Ciências da Computação.

Orientadora:

Profª. Drª. Simone das Graças Domingues Prado.

Aprovada em Janeiro, 2023.

Profª. Drª. Simone das Graças Domingues Prado - Orientadora
Departamento de Computação - Faculdade de Ciências - Unesp/Bauru

Profª. Drª. Márcia A. Zanoli Meira e Silva
Departamento de Computação - Faculdade de Ciências - Unesp/Bauru

Prof. Dr. Renê Pegoraro
Departamento de Computação - Faculdade de Ciências - Unesp/Bauru

Bauru, 20 de Janeiro de 2023

RESUMO

Dado um meio físico ou virtual, a busca por um caminho ótimo entre dois ou mais pontos é uma problemática muito estudada na computação, possuindo uma série de soluções já validadas, como o algoritmo de Dijkstra e A*. O problema é relevante para um vasto espectro de aplicações, incluindo tráfego de redes, planejamento robótico, simulações e jogos de computador. No entanto, cada aplicação possui um conjunto de requisitos e restrições particular, tornando necessária a adaptação de soluções a cada caso específico. No caso de malhas dinâmicas bidimensionais o algoritmo A* é comumente utilizado, fornecendo a solução de forma ágil e precisa, e sendo fácil de adaptar. Este trabalho traz uma análise teórica sobre o problema do caminho ótimo, partindo de uma perspectiva generalizada, a qual é, então, restringida a uma malha dinâmica bidimensional, a ser percorrida por um agente virtual, utilizando o algoritmo A*.

Palavras-chave: Caminho ótimo, Malha dinâmica, Dijkstra, A*.

ABSTRACT

Given a physical or virtual medium, the search for an optimal path between two or more points in that medium is a recurrent problem in computer science, with a series of already validated solutions, such as Dijkstra's algorithm and A*. The problem is relevant to a wide spectrum of applications, including network traffic, robotic planning, simulations and computer games. However, each application has a particular set of requirements and restrictions, making it necessary to adapt solutions to each specific case. In the case of two-dimensional dynamic meshes, the A* algorithm is commonly used, providing the solution in an agile and precise way, and being easy to adapt. This work presents a theoretical analysis of the optimal path problem, starting from a generalized perspective, which is then restricted to a two-dimensional dynamic mesh, to be traversed by a virtual agent, using the A* algorithm.

Keywords: Optimal path, Dynamic mesh, Dijkstra, A*.

SUMÁRIO

1 INTRODUÇÃO

Dado um meio físico ou virtual, a busca por um caminho ótimo que conecte dois ou mais pontos neste meio é uma problemática muito estudada na computação. Sua importância é refletida dentre diversas aplicações, incluindo tráfego de redes, planejamento robótico, simulações e jogos eletrônicos. Tipicamente, uma matriz é sobreposta sobre uma região, e uma busca por grafo é utilizada para encontrar o caminho. (YAP, 2002)

Em 1959 Dijkstra já propôs um algoritmo solução para resolver grafos desse tipo. Sua solução não utiliza conhecimento prévio sobre o problema, sendo, portanto, caracterizada como uma solução por força-bruta. No entanto, a utilização de conhecimentos adicionais sobre o espaço de problema permite grandes otimizações nas estratégias de busca de caminhos ótimos. Por exemplo, aplicar uma função heurística como estimativa da distância de cada vértice até o objetivo e utilizar seu valor para percorrer o grafo resulta em uma otimização do algoritmo de Dijkstra, sendo chamado de algoritmo A*. (LEWIS; DILL, 2015) (DIJKSTRA, 1959)

Além disso, para aplicações práticas sempre são necessários conjuntos particulares de requisitos e restrições adicionais, gerando novos desafios a serem superados. Essas restrições afetam desde a definição do espaço de problema até a aplicação da solução encontrada. Ainda mais, embora os grafos sejam tradicionalmente estudados como objetos estáticos, grafos dinâmicos representam melhores modelos do mundo real, onde mudanças na região acarretam na necessidade de adaptação das rotas traçadas. (MARTINS, 2012)

Este trabalho traz como proposta a análise da teoria fundamental por trás da busca por caminhos ótimos, iniciando com conceitos básicos sobre a modelagem de um espaço de problema e a quantificação da eficiência de algoritmos solução. Então serão apresentadas algumas das principais estratégias para lidar com casos mais genéricos de busca não informada e informada (quando se possui informação prévia sobre o espaço de problema), seguida de uma análise restringida a malhas dinâmicas bidimensionais, assim como Kuffner já propôs no final da década de noventa (FERREIRA, 2010), (KUFFNER, 1998). Como proposta prática, a malha deverá ser percorrida por um agente virtual, de forma análoga a casos comuns em jogos eletrônicos, onde geralmente se utiliza o algoritmo A*, sendo discutido, então, aspectos da implementação do algoritmo e desenvolvimento de um ambiente virtual de testes do mesmo, permitindo, por fim, uma demonstração prática da solução.

2 MODELAGEM DO PROBLEMA

O processo de resolução de problemas, muitas vezes, pode ser modelado como uma busca em um espaço de estados, a partir de algum dado estado inicial, e com regras que descrevem como transformar um estado em outro. Esse processo deve ser aplicado repetidamente até que satisfaça alguma condição objetivo. Nos casos mais comuns, visa-se o melhor desses caminhos, geralmente em termos de comprimento ou custo do caminho. (EDELKAMP; SCHRÖDL, 2012)

2.1 ESPAÇO DE PROBLEMA

O modelo de espaço de problema, formulado por Newell e Simon (1971), consiste em um conjunto de estados e um conjunto de operadores que mapeiam transformações entre os estados. Cada estado, ou configuração, é a representação abstrata do problema em um determinado instante. Um operador é uma ação aplicável sobre um estado, sendo considerado como um mapeamento parcial uma vez que podem haver restrições que determinem a quais estados pode ser aplicado. Um problema, ou instância de problema, é composto por um espaço de problema associado a um estado inicial e um conjunto de estados finais, ou estados alvo. O objetivo do problema é encontrar uma solução, isto é, uma sequência de operadores que, aplicados sobre o estado inicial do espaço de problema, irão mapeá-lo até um dos estados alvo. A solução encontrada pode ser avaliada em termos de custo, assim como espaço (memória) e tempo necessários para ser encontrada. (KORF, 1988) (NEWELL; SIMON, 1971)

O custo é uma medida conjunta de requisitos predeterminados a cada problema, e são associados à aplicação de cada operador. Determinar o custo de aplicação dos operadores é uma tarefa potencialmente complexa. Em casos simples, como no conhecido problema do caixeiro viajante, o custo representa o preço gasto para viajar até uma cidade. Porém, casos práticos devem ponderar uma série de fatores, de acordo com as particularidades da instância de problema e dos objetivos a que se deseja alcançar. Em casos de movimentação robótica, por exemplo, podem ser considerados aspectos como risco e dificuldade de uma rota. Para aplicações em redes de comunicação, o congestionamento de cada caminho será bastante relevante. Em aplicações de rotas de navegação por sistema de GPS (*Global Positioning System*), o tempo e a distância são importantes etc. Por serem específicos a cada caso, torna-se uma questão de interpretação do problema.

Os espaços de problema são comumente representados na forma de grafos.

Dessa forma, seja tomada a seguinte definição:

Definição 1.1 Um grafo de problema $G = (V, E, s, T)$ para o espaço de problema $P = (S, A, s, T)$ é definido tal que seja $V = S$ o conjunto de vértices, $s \in S$ o vértice inicial, T o conjunto de vértices objetivos, e $E \subseteq V \times V$ o conjunto de arestas que conectam os vértices, com $(u, v) \in E$ se e somente se existir um $a \in A$ com $a(u) = v$. (EDELKAMP; SCHRÖDL, 2012)

Assim, dado um determinado espaço de problema, tem-se que em sua representação na forma de grafo, cada vértice representa um determinado estado, com as arestas que os conectam representando os operadores aplicáveis sobre cada estado. Caso os operadores não possam ser aplicados de forma reversível, o grafo será um **grafo direcionado**. Caso seja possível, através da aplicação de um conjunto de operadores, regressar a um estado já visitado anteriormente durante a expansão do grafo, o mesmo será um **grafo cíclico**. Se o espaço de problema possuir um número finito de estados alcançáveis, o grafo será um **grafo finito**.

O espaço de problema pode ser um **espaço de custo uniforme** caso todos os operadores possuam o mesmo peso, ou, caso contrário, um **espaço de custo não uniforme**. Além disso, se ele possuir pesos negativos, será classificado como um **espaço de pesos negativos**.

Essas definições apresentadas implicam em resultados diferentes para as estratégias de busca, e sua distinção ao modelar um problema é essencial para se identificar qual a melhor abordagem na busca por uma solução.

2.2 COMPLEXIDADE

A complexidade do problema pode ser expressa em função de sua ramificação e profundidade da solução. A ramificação é definida como o número de estados novos que podem ser alcançados a partir da aplicação de um único operador sobre um estado prévio, normalizada para o espaço de problema, e tomada como constante para simplificar a análise. A profundidade da solução é definida como o menor número de estados necessários a se percorrer, entre o estado inicial e um estado alvo, isto é, o número de vértices expandidos para se atingir um estado alvo. O consumo de memória do problema será considerado como o número de estados que devem ser mantidos simultaneamente na memória do computador até se atingir o estado alvo. De forma similar, o consumo de tempo será considerado como o número total de estados expandidos pelo computador desde o estado inicial até se atingir um estado alvo. (KORF, 1988)

2.3 REPRESENTAÇÃO

Foram tomadas as definições para o espaço de problema, e seu conjunto de requisitos e restrições. A partir destas definições a representação técnica do problema será feita, de forma genérica, através de duas listas: a lista de vértices já explorados, chamada de lista fechada e a lista aberta, de vértices abertos mas ainda inexplorados, também chamados de fronteira da busca. (EDELKAMP; SCHRÖDL, 2012)

2.4 EXEMPLOS DE PROBLEMAS

Nesta seção, são apresentados dois exemplos de problemas que serão utilizados ao longo do texto para exemplificar as estratégias e variações dos algoritmos de busca.

2.4.1 Jogo do 15

O Jogo do 15 é um jogo de quebra-cabeças que foi popular em meados dos anos 80 a 90. Em sua versão mais comum, uma imagem é seccionada em 16 quadrados iguais e removida de um quadrado, geralmente o do canto direito inferior. Os 15 quadrados restantes são encaixados em um esquadrilho 4x4, de forma que fiquem fixados no mesmo, mas mantendo a posição 16 livre. Assim, é possível deslizar um dos quadrados adjacentes ao espaço livre, modificando sua posição. A imagem original é então embaralhada, através de diversas permutações consecutivas, e o objetivo do jogo é a reordenação da imagem original, através do ajuste da posição dos 15 quadrados no esquadrilho.

2.4.2 Cubo de Rubik

O Cubo de Rubik é um outro tipo de quebra-cabeças popular até hoje. Em sua forma original, um cubo é seccionado em 26 cubos menores, com cada face do cubo inicial possuindo 9 cubos, sendo que os cubos que formam as arestas e cantos do original são compartilhados por dois e três lados, respectivamente. Os cubos centrais a cada lado são conectados por eixos passando pelo centro do cubo original, e os demais são fixados entre si por encaixes, de forma a permitir sua rotação nos sentidos dos eixos cartesianos tridimensionais. Cada face do cubo possui uma cor, as quais são embaralhadas através de múltiplas rotações. O objetivo do jogo é a reordenação das cores dos 6 lados do Cubo de Rubik, através da rotação de suas peças.

3 ESTRATÉGIAS DE BUSCA

A seção anterior define a modelagem de uma instância de problema em um espaço de problema, sua representação na forma de grafo, seu conjunto de requisitos e restrições, e ainda outros conceitos necessários. Agora será dado início à análise de estratégias para a busca do caminho ótimo.

3.1 FORÇA-BRUTA

Considerando uma instância de problema, soluções de força-bruta (também chamadas de soluções desinformadas) nesse espaço são aquelas que não utilizam nenhum tipo de conhecimento prévio sobre o espaço de problema ou os estados alvo, sendo necessário percorrer sequencialmente os estados até encontrar uma solução. Nestas condições, as duas abordagens fundamentais são a busca por largura e a busca por profundidade. Porém, existem outras estratégias mais sofisticadas, como o algoritmo de Dijkstra, ou ainda a busca por aprofundamento iterativo.

3.1.1 Busca por largura

Numa busca por largura, os vértices de fronteira do espaço de problema são armazenados em uma lista do tipo FIFO (*first in first out*), isto é, numa fila onde os vértices serão analisados em ordem de adição. Isto traz duas consequências principais ao processo de busca. Primeiro, por ser uma busca horizontal no que diz respeito à profundidade de expansão dos vértices do grafo do espaço de problema, caso seja um problema de custo uniforme, a solução encontrada será sempre uma solução ótima. (EDELKAMP; SCHRÖDL, 2012)

Em segundo lugar, caso a ramificação do espaço de problema seja de grau elevado, o consumo de memória necessário para aplicar o algoritmo irá crescer rapidamente, com sua complexidade crescendo exponencialmente com o fator de ramificação do problema, o que pode exaurir os recursos de memória disponíveis para a busca antes de se encontrar uma solução. (KORF, 1988)

3.1.2 Busca por profundidade

Numa busca por profundidade, os vértices de fronteira do espaço de problema são armazenados em uma lista do tipo LIFO (*last in first out*), isto é, numa pilha onde se analisa o vértice mais recentemente adicionado na mesma. Como resultado, a busca

percorre o grafo expandindo seus vértices de forma vertical, obtendo resultados bem diferentes da busca por largura.

Caso o espaço de problema seja finito e acíclico, a busca por profundidade irá garantidamente encontrar uma solução, mas esta não será necessariamente uma solução ótima. (EDELKAMP; SCHRÖDL, 2012)

Além disso, o consumo de tempo necessário para encontrar a solução pode ser demasiadamente grande, dependendo da profundidade do grafo. Por outro lado, o consumo em memória da busca por profundidade é reduzido, crescendo linearmente com o fator de ramificação do problema. (KORF, 1988)

3.1.3 Algoritmo de Dijkstra

Em 1959, Dijkstra propôs uma solução para lidar com grafos de peso não uniforme, utilizando um algoritmo "guloso" de busca, que permite explorar os vértices do espaço de problema sequencialmente, e, desde que não hajam pesos negativos no mesmo, garante encontrar uma solução ótima. (LEWIS; DILL, 2015) (EDELKAMP; SCHRÖDL, 2012)

Sua estratégia é baseada no **princípio de otimalidade**, que diz que, num grafo, todo caminho ótimo do vértice A ao vértice B também será ótimo do vértice A ao vértice C, dado que C faça parte do caminho ótimo de A até B. Isto implica que, encontrar o caminho ótimo entre o vértice inicial e cada vértice intermediário antes do vértice final irá resultar no caminho ótimo até o vértice final. (EDELKAMP; SCHRÖDL, 2012).

Dessa forma, o algoritmo de Dijkstra irá efetuar uma varredura de todos os vértices a partir do vértice inicial, dando preferência sempre para o caminho de menor custo, e armazenando na lista fechada o custo até cada vértice já percorrido. Esse processo se repete até que o vértice alvo seja atingido, ou se extinguam as opções disponíveis na lista aberta. Caso seja alcançado, o caminho encontrado será um caminho ótimo. Por outro lado, caso o vértice alvo não seja atingido, significa que não existem caminhos possíveis que alcancem o estado final no grafo explorado.

3.1.4 Operadores com peso negativo

No contexto de busca não informada, o algoritmo de Dijkstra é uma ótima alternativa para grafos finitos de pesos não uniformes e positivos. Porém, para grafos onde haja a ocorrência de pesos negativos, o mesmo não mais garante que se encontre um caminho ótimo, ou mesmo que se encontre um caminho até o ponto final. Isto

porque o algoritmo de Dijkstra não permite que pontos adicionados à lista fechada sejam reacessados.

Neste caso, é possível adicionar uma pequena modificação de relaxamento no algoritmo, configurando um número limite de reanálises de cada vértice da lista fechada, amenizando este problema. Ainda assim, para grafos de peso negativo onde haja a ocorrência de ciclos com peso total negativo, passam a ser necessárias modificações maiores, resultando num novo algoritmo, o algoritmo de Bellman-Ford. (EDELKAMP; SCHRÖDL, 2012)

3.1.5 Aprofundamento iterativo

Outra fraqueza em potencial do algoritmo de Dijkstra é que o mesmo pode ser muito custoso em memória para grafos de tamanho extenso. (EDELKAMP; SCHRÖDL, 2012)

Uma alternativa para lidar como esse cenário é aplicar uma técnica chamada de aprofundamento iterativo sobre a busca por profundidade, resultando no algoritmo de busca por profundidade com aprofundamento iterativo (DFID), *depth first, iterative-deepening*. Neste caso, será aplicada a busca por profundidade conforme visto anteriormente. Porém, para cada etapa de aplicação do algoritmo será definida uma profundidade limite, que será progressivamente ampliada ao longo da busca.

O algoritmo irá buscar todos os caminhos atingidos com a aplicação de um operador sobre o ponto inicial no espaço de problema. Na sequência, o processo será repetido com a aplicação de dois operadores, aumentando progressivamente a profundidade limite, até que se encontre o ponto alvo. Dessa forma, o caminho obtido será um caminho ótimo, e o consumo em memória da busca será equivalente ao consumo de uma busca por profundidade. Por outro lado, embora pareça intuitivo que o consumo de tempo para se encontrar o caminho seja demasiado grande, considerando que o algoritmo percorre diversas vezes os mesmos caminhos, é possível comprovar matematicamente que ele será, no pior caso, uma ordem a mais que o consumo de tempo para uma busca por largura no mesmo espaço de problema. (KORF, 1988)

3.2 BUSCA HEURÍSTICA

Na seção anterior foram analisadas algumas das estratégias fundamentais de busca não informada. Quando não se tem conhecimento sobre o espaço de problema, as opções de busca se tornam limitadas. Por outro lado, quando existem informações prévias, como por exemplo, o grafo completo do espaço de busca, pode-se utilizar esse

conhecimento para obter boas otimizações no processo, convergindo mais rapidamente para o vértice alvo.

Existem diversas fontes de conhecimento que podem ser utilizadas na busca informada. Elas podem ser provenientes de conhecimento previamente adquirido ou ainda estimadas a partir de uma análise do espaço de problema. Em alguns quebra-cabeças populares, como o Cubo de Rubik, ou o Jogo do 15, pode-se estimar a distância de uma peça até sua posição final utilizando a distância de Manhattan, uma métrica de distância Euclidiana pelos eixos perpendiculares de movimento. Já em buscas de sistema de GPS, o próprio mapa da região, previamente conhecido, é utilizado.

De forma geral, uma função que estime a distância de um vértice no espaço de problema até o vértice alvo é chamada de função heurística, e sua adição ao algoritmo de Dijkstra resulta no algoritmo A*. (LEWIS; DILL, 2015)

Nas próximas seções serão analisados o algoritmo A* e algumas de suas variações. Na sequência, serão apresentadas algumas de suas limitações frente às necessidades no estado da arte de busca, no campo de jogos virtuais e outras aplicações, e, então, algumas propostas que têm sido avaliadas como promissoras para estes casos. Por fim, serão avaliados de forma superficial outros tipos de conhecimentos que também podem ser utilizados para otimizar a busca, como macros, abstrações e sub-objetivos.

3.2.1 A*

Menos de uma década após o surgimento do algoritmo de Dijkstra (1959), Peter Hart (1968), Nils Nilsson (1968) e Bertram Raphael (1968) do Instituto de Pesquisa de Stanford (atualmente SRI International), publicaram uma alternativa otimizada, popularizada como algoritmo A*. (LEWIS; DILL, 2015) (HART; NILSSON; RAPHAEL, 1968)

Esse algoritmo trouxe como proposta a aplicação de uma função como estimativa do custo de cada vértice até o vértice alvo, e a utilização dessa função para guiar a ordem de expansão dos vértices no grafo pelo algoritmo de busca.

A função de estimativa, nomeada função heurística, pode ser definida como uma função de avaliação, que mapeia o peso de cada vértice até o vértice objetivo no espaço de problema. Essa função será admissível caso os pesos avaliados nunca sejam maiores do que os pesos reais conectando os vértices avaliados e o objetivo. (EDELKAMP; SCHRÖDL, 2012)

Desta forma, seja $f(x)$ a função de avaliação de custo para cada vértice do grafo; $g(x)$ o custo real do vértice e $h(x)$ o custo estimado pela função heurística para o mesmo. Então tem-se que $f(x) = g(x) + h(x)$, e, caso $h(x)$ seja uma função heurística admissível, o caminho encontrado aplicando $f(x)$ será um caminho ótimo. (KORF, 1988)

3.2.2 A* com aprofundamento iterativo

Por ser uma otimização do algoritmo de Dijkstra, o algoritmo A* também está sujeito às mesmas fraquezas e limitações de seu predecessor. Assim, espaços de problema muito custosos em memória podem ser um desafio para ele. Mas, por outro lado, pode-se utilizar a mesma estratégia aplicável ao algoritmo de Dijkstra para superar este problema, resultando no algoritmo A* com aprofundamento iterativo, ou IDA* (do inglês *Iterative Deepening A**).

O IDA* é análogo ao A*, mas a expansão dos vértices será feita considerando um limiar pré definido de custo estimado em cada iteração. Este custo será calculado a partir da menor estimativa do vértice atual, e aumentado gradualmente de forma a sempre permitir a expansão de ao menos um vértice. Assim, o algoritmo irá trocar parte do consumo de memória por tempo, e irá parar assim que encontrar uma solução, que será garantidamente uma solução ótima. Caso ele extinga a lista aberta sem encontrar uma solução, será por que não há uma solução possível. (EDELKAMP; SCHRÖDL, 2012)

Um outro aspecto importante do IDA* é que, em espaços de problema cíclicos, se torna essencial garantir que o algoritmo expanda apenas uma vez cada vértice, ou caso contrário sua complexidade irá crescer de forma quadrática. (EDELKAMP; SCHRÖDL, 2012)

3.2.3 Limitações e alternativas ao A*

Em alguns tipos de espaço de problema, nos quais múltiplos agentes são considerados de forma simultânea, o A* e mesmo o IDA* ainda podem se tornar muito custosos em memória, pois deverão avaliar de forma individual o caminho de cada agente. Casos assim tem se tornado cada vez mais comuns em jogos eletrônicos, entre outras aplicações, e, por conta disso, tem estimulado o desenvolvimento de alternativas ao A* que possam superar tais dificuldades. (FOEAD et al., 2020)

A principal estratégia de otimização, que pode ser usada como uma primeira abordagem a este tipo de problema, é a otimização do mapeamento do espaço de problema. Por exemplo, em alguns casos é viável pré-calcular cada caminho possível

para os agentes, e mantê-los em memória. Assim, o número de agentes pode ser escalado sem aumento no custo de espaço e com um baixo aumento no custo de tempo. Mas, mesmo em casos que isto não seja possível, buscar otimizações no mapeamento trarão os melhores resultados na redução dos custos da busca multi-agente. (RABIN; STURTEVANT, 2014)

Ainda assim, existem casos em que pode não ser possível otimizar o mapeamento, sendo necessário utilizar outras abordagens de otimização. Uma alternativa ao algoritmo A* e suas variantes para lidar com a busca de caminhos ótimos multi-agente é a utilização de campos de potencial. Nessa estratégia, o foco não está em buscar o caminho ótimo a partir da posição de cada agente até o vértice alvo, mas sim em mapear para cada vértice, qual a direção ótima até o vértice alvo. O campo de potencial é uma aproximação de uma **função de fluxo** contínua, aplicada a cada vértice do grafo do espaço de problema, e mantida em memória, de forma que, cada agente necessite apenas consultar qual o sentido de movimento ótimo de sua posição atual para o vértice alvo. Desta forma, cada agente irá fazer apenas uma consulta simples sobre cada vértice do caminho, sendo possível escalar o número de agentes com um baixo aumento no custo de tempo do problema. (PENTHENY, 2014)

3.2.4 Outras Otimizações

Existem diversas formas de se aplicar o algoritmo A*, de acordo com as necessidades da busca. No entanto, a função heurística não é a única maneira de se utilizar conhecimentos prévios como fonte de otimização. Existem outras estratégias aplicáveis que modificam o espaço de problema e trazem vantagens na busca, como a utilização de macros, abstrações e sub-objetivos.

3.2.4.1 Macros

Em alguns espaços de problema, como no Jogo do 15 ou no Cubo de Rubik, o caminho até a solução não é uma sequência de operadores que se aproximam linearmente até o vértice alvo do grafo. Nestes exemplos é necessário aplicar operadores que irão afastar o estado do espaço de problema de sua solução, a fim de permitir que se alcance a mesma. E, quanto mais próximo da solução, maior a inevitabilidade desses "passos para trás".

Essa caraterística pode ser custosa para os algoritmos de busca lidarem, pois estes geralmente procuram uma aproximação progressiva até o vértice solução. Tomando o Jogo do 15 como exemplo, para se posicionar corretamente cada quadrado no esquadrilho é necessário uma sequência de operadores, constante durante o espaço

de problema para cada peça, e que irá afastar o estado do grafo ao vértice solução antes de avançar, com o posicionamento correto da peça.

Como essa sequência é constante (e simétrica entre suas variações de sentido de rotação da peça), ela pode ser aprendida nas etapas iniciais da solução do problema, e, então, convertida num macro de operadores, isto é, um conjunto sequencial de operadores que possam ser lidados pelo algoritmo solução como um único operador. Dessa forma, a utilização do macro como operador evita o afastamento da solução durante a busca, permitindo que a mesma seja feita de maneira progressiva, facilitando o processo e otimizando o algoritmo utilizado. (KORF, 1988)

3.2.4.2 Abstrações

Em outros casos, como na busca por sistema de GPS, o espaço de problema é demasiadamente grande e complexo para ser analisado integralmente na busca por uma solução quando a rota que se deseja for entre dois vértices distantes. Considere um usuário do sistema que deseje viajar de carro entre duas cidades muito distantes. De fato, não é necessário considerar cada interseção de caminhos em todas as cidades intermediárias entre o vértice de origem e o vértice destino, pois a conexão entre estes será dada por uma rodovia.

Dessa forma, o algoritmo de busca pode abstrair o espaço de problema em níveis de detalhe, ou aproximação, considerando a rota que interligue os vértices origem e destino à rodovia, utilizando o mapa do espaço de problema em sua forma mais detalhada para tal, mas abstraindo o mesmo para um nível de afastamento que considere apenas as rodovias e estradas, e, portanto, seja muito mais simples de computar para traçar a rota interurbana.

Assim, classificar as arestas do espaço de problema em níveis de abstração permite controlar o volume de possibilidades relevantes em diferentes estágios da busca, garantindo uma redução nos recursos necessários para que o algoritmo calcule as soluções possíveis. (KORF, 1988)

3.2.4.3 Sub-objetivos

Temos ainda, casos em que a ramificação de cada vértice no espaço de problema é muito elevada, mas, para se atingir a solução é necessário percorrer certos vértices prédeterminados. Por exemplo, no problema do Cubo de Rubik, antes de se solucionar todos os lados do cubo é necessário solucionar um lado inicial. Como todos os lados possuem as mesmas propriedades, não importa qual lado seja escolhido. Este estado do espaço de problema com um lado solucionado pode ser considerado como

um sub-objetivo da solução. Assim, definir sub-objetivos permite fracionar o problema, reduzindo os recursos de memória necessários para se buscar a solução. (KORF, 1988)

4 MALHA DINÂMICA

No capítulo anterior foram analisadas diversas estratégias de busca, partindo de casos fundamentais, como a busca desinformada sobre grafos de peso uniforme, grafos não uniformes e ainda grafos com peso negativo, chegando então em buscas informadas e na utilização de funções heurísticas ou outras aplicações de conhecimento prévio para otimizar o processo. Esta seção irá avançar para a análise de um caso particular de aplicação prática dos métodos de busca.

Seja o espaço de problema uma matriz bidimensional, **N** x **M**, de peso uniforme, e com obstáculos dinâmicos, isto é, os vértices do espaço de problema serão bloqueados e liberados de forma dinâmica durante a execução da busca.

Existe uma distinção entre encontrar um caminho ótimo e navegar por esse caminho. A navegação pelo caminho encontrado envolve fatores adicionais a se considerar, como o tempo de deslocamento do agente. Em casos estáticos, onde não há alteração do espaço modelado, isso não gera um problema, pois independentemente do tempo gasto, a solução permanece constante.

Porém, em casos dinâmicos, tanto pela modificação do meio como pela presença de múltiplos agentes, a navegação pelo caminho se torna um desafio.

Para analisar o processo de percorrimento do caminho encontrado, é necessário considerar o tempo de deslocamento do agente. Isso reflete tanto situações comuns em jogos eletrônicos, onde os agentes devem percorrer os caminhos em tempo análogo ao de jogadores humanos, como na aplicação em robótica, onde limitações físicas resultam em tempos de percorrimento muito longos se comparados aos ciclos de processamento de computadores modernos.

Passa a ser necessário, então, manter um controle da posição atual do agente que percorre o caminho, relativa ao mesmo, e, caso haja alterações no grafo, o caminho deve ser reavaliado, uma vez que o mesmo pode ter se tornado impossível, ou pode haver uma nova rota melhor do que a atual.

4.1 ALGORITMO

Definido o espaço de problema, um grafo finito, não direcionado e cíclico, de custo uniforme e sem pesos negativos, será utilizado o algoritmo A* em sua forma mais

básica para buscar a solução. Conforme descrito anteriormente, serão utilizadas duas listas para armazenar os vértices de fronteira e os vértices já visitados, chamadas de lista aberta e fechada respectivamente.

O peso de cada vértice da matriz do espaço de problema será calculado a partir de uma função heurística de distância, que irá considerar a distância linear do vértice avaliado até o vértice objetivo. Os vértices adicionados na lista aberta serão ordenados a partir dessa estimativa de distância, e analisados em ordem de prioridade da menor distância.

A lista fechada não irá permitir a revisitação dos vértices, sendo que cada um será avaliado uma vez por execução da busca. Caso o espaço de problema sofra alterações, seja nos vértices intermediários, ou na posição do vértice alvo, o algorítimo de busca será reexecutado para cada agente virtual, uma vez que o mesmo representa um meio dinâmico.

O caminho calculado para cada agente será armazenado em uma terceira lista. Caso esse caminho alcance o vértice objetivo, o respectivo agente irá percorrer o caminho, vértice a vértice, até que atinja o objetivo, com os vértices percorridos sendo removidos da lista do caminho ótimo calculado.

De acordo com a documentação fornecida com o pacote utilizado, se estima um consumo de memória de 50 Mb para um espaço de problema de ordem 1024 x 1024. No caso deste trabalho, será utilizada uma matriz bem menor, na ordem de 17 x 9, pois permite uma visualização mais clara do processo de busca. Desta forma, se estima que o gastos de memória para o cálculo do caminho ótimo serão menores que 50 Mb, sendo pequenos o bastante para serem desconsiderados.

5 DESENVOLVIMENTO

A parte prática deste trabalho foi elaborada considerando os estudos apresentados nos capítulos anteriores. O objetivo almejado foi o de apresentar uma aplicação do algoritmo A* num ambiente dinâmico e controlável. Seguem os detalhes técnicos do desenvolvimento.

5.1 TECNOLOGIAS

Como principal ferramenta de desenvolvimento foi utilizado o motor de jogos Unity, com código desenvolvido em C#. O Unity é uma ferramenta poderosa e versátil, de uso gratuito para estudantes e usuários que não estejam atuando em capacidade profissional. Ele fornece uma série de bibliotecas orientadas ao desenvolvimento de jogos eletrônicos, com portabilidade multi-plataforma e uma ampla documentação sobre seu uso. A linguagem C# é a linguagem oficial de desenvolvimento para o Unity, sendo diretamente compatível com sua utilização. (UNITY, 2022)

Além das ferramentas orientadas a jogos, o Unity também traz uma gama extensa de pacotes de terceiros, com os mais diversos propósitos. Para este trabalho foi utilizado o pacote "*2D Grid Based A* Pathfinding*", de autoria de Saad Khawaja, e utilizando uma solução para o A* por Christoph Husse, com uma fila de prioridade por Gustavo Franco. (KHAWAJA; HUSSE; FRANCO, 2015). O pacote é de uso livre e gratuito, e traz uma implementação do A* adaptada a uma matriz de **N x M** vértices. Esta implementação considera que os vértices podem ser bloqueados e desbloqueados, sendo compatível com o propósito deste trabalho, para uma malha dinâmica. Além disso, o algoritmo utiliza uma fila de prioridade, que atua ordenando os vértices adicionados na lista aberta. Como o caso particular deste trabalho considera um espaço de problema de **pesos uniformes**, a única métrica relevante para ordenação dos vértices é o valor estimado na função heurística, ou seja, a distância de cada vértice ao vértice objetivo.

Por fim, o pacote também traz uma interface que apresenta o funcionamento do algoritmo em um ambiente dinâmico, mas sem permitir a alteração do vértice alvo. Por conta disso, e, para melhor representar um ambiente dinâmico onde, tanto o espaço de problema, quanto a posição do vértice alvo possam sofrer alterações durante o trajeto do agente virtual, a interface fornecida não foi utilizada, sendo substituída por uma interface desenvolvida pelo autor especificamente para este trabalho.

Também foram utilizados alguns *assets*, para a representação visual dos ele-

mentos da aplicação, como os agentes virtuais, o vértice objetivo e a obstrução dos vértices do espaço de problema. Os *assets* utilizados são provenientes do jogo *Glitch*, um jogo multi-jogador publicado em 2009, mas que foi encerrado em 2012. Após o encerramento do jogo, todos os *assets* foram disponibilizados para uso livre e gratuito, e podem ser acessados e baixados através do link: www.glitchthegame.com.

5.2 METODOLOGIA

A primeira etapa no desenvolvimento da demonstração prática deste trabalho foi a busca por soluções já existentes, a fim de se encontrar boas referências de uso para o caso particular de interesse em questão. Após um breve período de pesquisa, foi encontrada uma solução promissora, como parte do pacote *2D Grid Based A* Pathfinding*, descrito anteriormente. Na sequência realizou-se o estudo e análise da solução do A* importada para que se pudesse compreender o suficiente das interfaces de implementação a fim de utilizá-las neste trabalho.

De forma simplificada, para utilizar o A* a partir do pacote selecionado é necessário definir um vetor de vértices **N x M**, que estenda a interface de vértices fornecida, nomeada *TPathNode*. Esta interface contém a implementação de uma função para verificar se o vértice está bloqueado ou não. Após definido este vetor, é necessário implementar uma classe nomeada *My Solver*, que irá receber o vetor **N x M** definido anteriormente. Por fim, basta chamar a função interna do *My Solver*, nomeada *Search*, que irá receber por parâmetro os vértices inicial e final da busca que se deseja realizar. A função *Search* também possibilita a adição de um contexto, que permite passar parâmetros adicionais para o A* caso a utilização desejada tenha tais necessidades. Para este trabalho apenas os parâmetros dos vértices foram necessários, sendo mantido o parâmetro de contexto como nulo, uma vez que as funcionalidades utilizadas não necessitaram do mesmo.

Após esse período foi escrito um algoritmo A*, a fim de avaliar algumas outras possibilidades em sua implementação e interface de utilização, além de permitir um estudo mais aprofundado do mesmo. Porém, pouco foi feito de forma significativamente diferente da solução já disponível. Por outro lado, a implementação da fila de prioridade se mostrou uma tarefa potencialmente extensa, de forma que se considerou apropriado utilizar a fila de prioridades já fornecida. Como consequência desta decisão, foram necessários alguns ajustes na solução em desenvolvimento para o A* que o tornaram ainda mais similar à solução já disponível.

Portanto, foi concluído que a melhor estratégia seria a utilização direta do A* conforme disponível, com o desenvolvimento apenas da interface de utilização e dos

elementos gráficos, a fim de evitar retrabalho e desenvolvimento redundante, mas obtendo o máximo de flexibilidade em sua utilização. A reescrita da interface gráfica permitiu a adaptação desejada do algoritmo solução para uma malha dinâmica que também considera a modificação da posição do vértice alvo.

Os detalhes técnicos do algoritmo A* utilizado já foram apresentados anteriormente, na seção 4.1.

Por fim, foi feito o desenvolvimento da interface gráfica, e cujos resultados são apresentados a seguir.

5.3 RESULTADOS

A aplicação foi desenvolvida com sucesso, permitindo a demonstração do funcionamento do A* em um ambiente de grade bidimensional dinâmica, de forma que possa ser simulada tanto a obstrução e desobstrução do caminho entre o agente virtual e seu objetivo, como ainda a movimentação do objetivo na grade do espaço de problema.

Na aplicação, cada agente virtual recebe uma instância do algoritmo de busca, realizando sua própria busca até o vértice objetivo, que poderia ser diferente para cada agente, embora neste trabalho tenha sido mantido um mesmo vértice, a fim de simplificar a demonstração dos resultados.

Além disso, foi desenvolvida uma representação visual do caminho traçado por cada agente virtual, atualizada dinamicamente conforme a malha sofre alterações, e que mostra a contagem de vértices em ordem decrescente até o vértice objetivo, que recebe a posição zero.

Também foi implementada a geração de múltiplos agentes virtuais de forma simultânea, permitindo a visualização de uma busca multi-agente, embora, não tenham sido desenvolvidas formas de interação entre os agentes.

Uma vez iniciada a aplicação, é instanciado um agente virtual que efetua a busca do vértice alvo no espaço de problema, começando a se deslocar em sua direção assim que a busca retornar com sucesso.

A atualização dos vértices da malha do espaço de problema é feita a partir de cliques do mouse em cada vértice na tela, alternando-os entre bloqueados ou desbloqueados, durante o processo de busca. Também é possível o ajuste da posição do vértice alvo através do arraste de seu indicador visual com o mouse. A instanciação

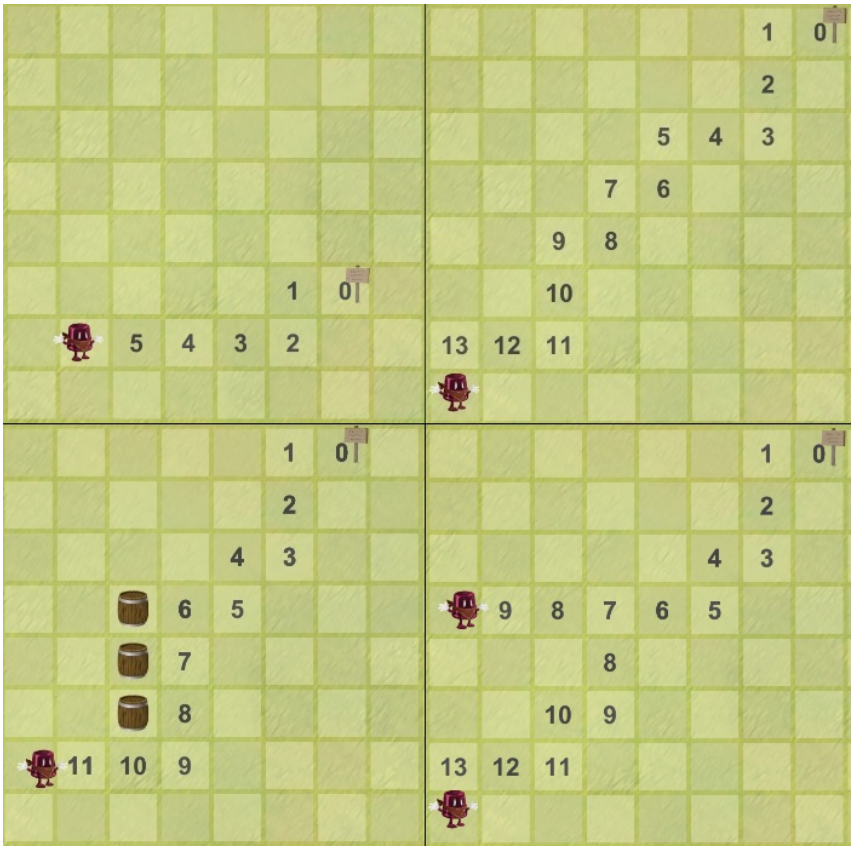
de novos agentes virtuais pode ser feita pela barra de espaço do teclado, com um agente novo sendo instanciado a cada acionamento da tecla.

Para cada agente, o processo de busca é efetuado de forma independente. Uma vez atingido o vértice alvo, o agente sinaliza o sucesso de sua trajetória através de uma mensagem pelo terminal, sendo destruído na sequência. Sempre que a malha do espaço de problemas sofre qualquer alteração, seja na modificação da obstrução de alguns de seus vértices, ou no ajuste da posição do vértice objetivo, o algoritmo de busca é reexecutado para cada agente. Para identificar as alterações na malha foi implementada uma arquitetura de eventos, na qual cada agente virtual é inscrito ao ser instanciado, e que tem como gatilho os cliques do mouse nos vértices da tela e a ativação da tecla de espaço. Sempre que for reconhecido um gatilho, o sistema notifica cada agente inscrito, que irá, então, reexecutar o algoritmo de busca.

Conforme discorrido na seção 4.1, neste trabalho se estima um custo inferior a 50 Mb para o algoritmo de busca, de forma que o mesmo possa ser desconsiderado, mesmo que para múltiplos agentes agindo de forma simultânea.

Na figura 1 abaixo, as imagens apresentam da esquerda para direita e de cima para baixo: Um caminho ótimo com 6 vértices; um caminho ótimo com 14 vértices; um caminho ótimo calculado após obstrução do caminho anterior; e os caminhos ótimos calculados simultaneamente por dois agentes distintos.

Figura 1 – Demonstração visual da previsão de caminho ótimo calculado do agente virtual até o ponto objetivo.



Fonte: Imagem gerada a partir do aplicativo de demonstração do caminho ótimo apresentado neste trabalho. Criação do autor.

6 CONCLUSÃO

A busca pelo caminho ótimo é um problema muito versátil, tanto em suas aplicações como na proposta de soluções. Estudado desde antes da década de 60, diversas soluções já foram apresentadas. Partindo do caso inicial de busca não informada, as soluções mais fundamentais são a de busca por largura e busca por profundidade, mas existem outras abordagens, como o algoritmo de Dijkstra e a busca por aprofundamento iterativo. Cada técnica traz um conjunto de qualidades e fraquezas que devem ser consideradas para se encontrar a melhor solução a cada caso específico.

Uma importante maneira de otimizar o processo de busca pelo caminho ótimo é a utilização de conhecimentos prévios, aplicados no algoritmo de busca. Utilizar uma estimativa da distância do vértice objetivo a cada vértice do grafo de busca é uma técnica chamada de busca heurística, e associada ao algoritmo de Dijkstra resulta no algoritmo A*, um dos algoritmos de busca mais populares da busca informada. Embora muito popular, o A* apresenta perda de desempenho quando o objetivo é a busca ótima multi-agente, sendo nesse caso analisada a utilização da busca por campos de potencial como alternativa ao mesmo. Além da utilização da função heurística, existem outras aplicações de conhecimento que podem ser utilizadas para obter otimizações na busca, como a utilização de macros, abstrações e sub-objetivos.

Outro aspecto a se considerar na busca é o processo de navegação da rota pelos respectivos agentes, uma etapa que pode trazer complexidade adicional.

Este trabalho utiliza os conhecimentos teóricos apresentados para elaborar uma demonstração prática do algoritmo A*, através da utilização de uma biblioteca que traz uma implementação do mesmo, complementada com o desenvolvimento de funções auxiliares e uma interface visual. A demonstração considera o caso particular de uma malha dinâmica bidimensional, percorrida por múltiplos agentes virtuais utilizando o algoritmo, com uma representação visual do processo de busca e da atualização do caminho conforme o grafo do espaço de problema sofre alterações, e com uma arquitetura de eventos para otimizar o processo de busca.

Como continuidade deste trabalho é proposta a adição de novas características ao processo de visualização, como a interação entre agentes, a fim de permitir a comparação de outros métodos de busca, com interesse especial na busca multiagente por campos de potencial.

REFERÊNCIAS

DIJKSTRA, E. W. A note on two problems in connexion with graphs. Numerische Mathematik, v. 1, p. 269–271, 1959.

EDELKAMP, S.; SCHRÖDL, S. **Heuristic Search, Theory and Aplications**. 225 Wyman Street, Waltham, MA 02451, USA: Morgan Kaufmann, Elsevier, 2012. v. 1.0.

FERREIRA, J. A. S. Implementação e teste de um algoritmo planejador de caminhos em um jogo de estratégia de tempo real. Universidade Federal do Rio Grande do Sul, Porto Alegre, 2010.

FOEAD, D.; GHIFARI, A.; KUSUMA, M. B.; HANAFIAH, N.; GUNAWAN, E. A systematic literature review of a* pathfinding. In: **5th International Conference on Computer Science and Computational Intelligence 2020**. [S.l.]: Procedia Computer Science - Elsevier, 2020.

HART, P. E.; NILSSON, N.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE transactions of systems science and cybernetics**, ssc-4, Nº 2, julho 1968.

KHAWAJA, S.; HUSSE, C.; FRANCO, G. **2D Grid Based A* Pathfinding**. 2015. Pacote desenvolvido para Unity™, v. 1.2. Acesso em 30 de novembro de 2022. Disponível em: <https://assetstore.unity.com/packages/tools/ai/2d-grid-based-a-pathfinding-tower-defense-20087>.

KORF, E. R. Optimal path-finding algorithms. **Symbolic Computation - Search in Artificial Intelligence**, University of Portsmouth Southsea, Hants, UK, v. 1.0, 1988.

KUFFNER, J. J. J. Goal-directed navigation for animated characters using real-time path planning and control. In: THALMANN, N.; THALMANN, D. (Ed.). **Modelling and Motion Capture Techniques for Virtual Environments**. Stanford, CA 94305-9010, USA: Springer, Berlin, Heidelberg, 1998. (Lecture Notes in Computer Science, v. 1537).

LEWIS, M.; DILL, K. Game ai appreciation, revisited. In: RABIN, S. (Ed.). **Game AI Pro 2**. 6000 Broken Sound Parkway NW, Suite 300: CRC Press, Taylor Francis Group, 2015. v. 1.0.

MARTINS, G. A. V. Manutenção de caminhos mínimos em grafos dinâmicos. Universidade Federal do Paraná, Curitiba, 2012.

NEWELL, A.; SIMON, H. A. Human problem solving: The state of the theory in 1970. **American Psychologist**, American Psychological Association, v. 26(2), fevereiro 1971.

PENTHENY, G. Efficient crowd simulation for mobile games. In: RABIN, S. (Ed.). **Game AI Pro**. 6000 Broken Sound Parkway NW, Suite 300: CRC Press, Taylor Francis Group, 2014. v. 1.0.

RABIN, S.; STURTEVANT, N. R. Pathfinding architecture optimizations. In: RABIN, S. (Ed.). **Game AI Pro**. 6000 Broken Sound Parkway NW, Suite 300: CRC Press, Taylor Francis Group, 2014. v. 1.0.

UNITY. **Unity ©**. Unity Technologies, 2022. Acesso em 30 de novembro de 2022. Disponível em: <https://unity.com/>.

YAP, P. Grid-based path-finding. **Advances in Artificial Intelligence, Canadian AI 2002 Lecture Notes in Computer Science**, Springer, Berlin, Heidelberg, v. 2338, 2002.