Simulador de Braço Robótico guiado por algoritmos da Inteligência Artificial

Thiago Galves Moretto[1]

[1]CCET - Centro de Ciências Exatas e da Terra
Universidade Católica Dom Bosco
Av. Tamandaré, 6000, Jardim Seminário
79117-900 Campo Grande, MS

thiago@moretto.eng.br

Resumo. *Este trabalho aborda o desenvolvimento, testes e resultados de um simulador de braço robótico guiado até um alvo utilizando algoritmos de Inteligência Artificial. Avaliamos a eficiência de vários algoritmos de busca e discutimos os resultados.*

1. Introdução

Simular computacionalmente a capacidade humana de resolver problemas é a área de pesquisa denominada Inteligência Artificial. O inicio do desenvolvimento desta área surgiu logo após a Segunda Guerra Mundial através do artigo *Computing Machinery and Intelligence* de Alan Turing [Turing, 1950]. Desde então as pesquisas feitas na área são utilizadas e outras, como por exemplo, em Jogos[Funge, 2004], em Visão Computacional[Forsyth and Ponce, 2002] e Robótica[Ferguson et al., 2005].

Este trabalho apresenta uma implementação de um simulador de braço robótico utilizando algoritmos desenvolvidos pela área de Inteligência Artificial. O problema para ser solucionado é realizar movimentos com o braço robótico para que este atinga um pré-determinado alvo e um espaço virtual 3D. Os algoritmos propostos têm que encontrar uma sequência de movimentos que leve até esta meta.

A organização deste texto esta da seguinte forma, na seção 2 abordamos os trabalhos correlatos. Na seção 3 resenhamos uma pequena introdução dos algoritmos utilizados. Na seção 4 apresentamos detalhes da implementação nesta fase do trabalho. Na seção 5 apresentamos experimentos realizados e por fim na seção 6 temos as considerações finais.

2. Trabalhos Correlatos

Os trabalhos a seguir foram escolhidos por utilizarem algoritmos avaliados neste trabalho.

Em [Yun and Park, 2006], é proposto uma técnica de otimização estocástica, utilizando três métodos, entre eles, *Simulated Annealing*, para otimizar o tráfego sobre semáforos em um corredor urbano. [1].

Em [Vaziri et al., 2005], os autores propõem uma forma de alocar e agendar recursos e tarefas (de duração incerta) em vários projetos de maneira eficiente. Para isto, utilizam o método *Simulated Annealing* para achar a melhor forma de alocar os recursos,

[1]Disponível no site: http://mctrans.ce.ufl.edu/featured/TSIS/Version5/corsim.htm

e que estes não impençam, o mínimo possível, de que outras tarefas sejam executadas por outros projetos, ou seja, busca um melhor equilíbrio entre projetos, recursos e tarefas.

Uma variação do algoritmo *Hill Climbing* e o método *Simulated Annealing*, entre outros, é apresentada em [Xi et al., 2004] para otimizar configurações de servidores de aplicações. Este trabalho apresente a implementação destes algoritmos para encontrar uma melhor configuração dos vários parâmetros que os servidores de aplicações possuem, por exemplo, memória, *pool* de *Threads*, tamanho de *cache*, *timeouts*, etc.

Em [Korf, 1995], Korf expõe o problema do espaço em memória utilizado em métodos para satisfação de restrições e otimização em jogos por busca heurística e espaço de estados. Explica os problemas dos métodos *A**, da busca em largura e da busca em profundidade, propõe então como solução a utilização do *Iterative Deepening Search* (Busca por aprofundamento iterativo).

3. Algoritmos de busca

Nesta seção apresentaremos uma breve introdução dos algoritmos de busca implementados nesta fase do trabalho. Estes são divididos em três tipos, buscas exaustivas, buscas heurísticas e buscas locais.

3.1. Busca em Largura

A busca em largura consiste em uma busca em árvore. Inicialmente visitando a raíz e todos os seus vizinhos. Após isto, o mesmo processo é feito com os vizinhos, suscetivamente.

A busca em largura não utiliza nenhuma heurística para definir o melhor caminho a seguir até a sua meta, por isto é chamada de busca exaustiva, já que visita todos os nós possíveis até encontrar a solução.

A sua variação com limitação, Busca em Largura Limitada, surge para limitar a busca e retonar falha caso não encontre a solução dentro desse limite.

3.2. Busca em Profundidade e Aprofundamento Iterativo

A Busca em Profundidade explora todo um ramo de uma árvore até retornar e explorar o ramo vizinho. Uma vantagem deste método é encontrar o menor caminho do nó inicial até o nó final mais próximo. Entretanto, não garante a solução mais próxima. Em problemas onde há um grande número de soluções ou se a maioria dos caminhos puder levar à uma solucão este método pode ser eficiente.

A busca por aprofundamento iterativo é basicamente a busca em profundidade com dois mecanismos a mais, um limitador de profundidade e outro que incrementa este limite sempre que toda a árvore dentro do limite atual é explorada. Uma vantagem desta busca é poder encontrar a melhor solução.

3.3. Greedy Search

Greedy Search, ou busca gulosa, é um tipo de busca *best-first*, que refere-se em sempre prosseguir pelo caminho onde uma heurística informa ser melhor, desta forma, este algoritmo pode não chegar na meta, já que mesmo com uma heurística, as informações que esta passa não garantem que se chegue à meta [Russell and Norvig, 1995].

3.4. Beam Search

A *Beam Search* é uma otimização do algoritmo *best-first*, a diferença é que este algoritmo apenas expande os *m* melhores nós a cada iteração. Chamado de *beam width*, este valor

faz com que sejam visitados apenas àqueles nós que podem oferecer melhores caminhos até à meta.

3.5. A-*

Descrito em 1968 por Peter Hart, Nils Nilsson, e Bertram Raphael[Hart et al., 1972], o *A** é um algoritmo de busca informada em grafo, derivada do *best-first*, que busca o melhor caminho de um nó inicial até um nó meta. Para ser eficiente é necessário uma heurística admissível que define através de um valor qual melhor caminho à seguir, além disto, associa ao valor da heurística o custo atual do caminho, isto faz com que o algoritmo teste outra alternativas mesmo que a heurística informe que há outro caminho mais próximo da meta.

3.6. Gradient Descent

O *Gradient Descent* é uma busca local que tem o objetivo de minizar uma função. Esta busca sempre escolhe o melhor ou o primeiro melhor nó para expandir e não apenas

3.7. *Simulated Annealing* ou Têmpera Simulada

Simulated Annealing é um método de busca estocástico análogo ao processo físico de resfriamento gradativo para a produção de cristais utilizando o mínimo de energia possível. O SA tentar evitar os mínimos locais e tem como objetivo a busca do mínimo global [Carson and Maria, 1997].

4. Implementação

A implementação do simulador foi feita utilizando a linguagem Java, para aproveitar a plataforma, que dá suporte à utilização do simulador em vários sistemas operacionais, e para a utilização da biblioteca *Java 3D*, que disponibiliza uma API de alto-nível para a elaboração de ambientes 3D com suporte a vários tipos de objetos, comportamentos, iluminação, entre outras características prontas.

A parte gráfica nesta fase do trabalho é simples, consiste em um menu, que ainda esta em implementação, com as opções de algoritmos a serem utilizados para resolver o problema do deslocamento do braço robótico, estes algoritmos não estão disponíveis no menu. Há uma implementação em 3D de um homem (tronco, cabeça e membros superiores), com possibilidade de utilizar o mouse para rotacionar o ambiente.

Na **Figura 1** temos o diagrama de classes que apresenta a modelagem utilizada no simulador. A interface *Search* é implementada pelos algoritmos de busca, que deve, ao término da busca, retornar o estado final, ou o valor nulo, caso haja um fracasso na busca.

A interface *State* é uma representação básica de um estado, e de acordo com cada problema ele pode ser extendido ou implementado de qualquer forma, desde que, implemente dois métodos, o *parent* (Para ser possível realizar o caminho inverso) e o *equals* (para comparar os estados).

Para o algoritmo *SimulatedAnnealing* o modelo de estado (*State*) é inadequado, é necessário que este modelo tenha informações relacionadas ao valor do estado (assumindo x como o estado e y, o valor deste estado), ou seja, cada estado precisa de um valor, da mesma forma que em uma função $y = f(x)$, cada valor de x deva ter um valor y correspondente - a imagem de x pela função f. Para suprir essa necessidade, o modelo *State* foi extendido pela classe abstrata *SimulatedAnnealingState*, que implementa os métodos descritos pela interface *State* e define dois novos métodos que devem ser implementados (métodos abstratos), *getY* e *produce* (a assinatura pode ser vista na **Figura**

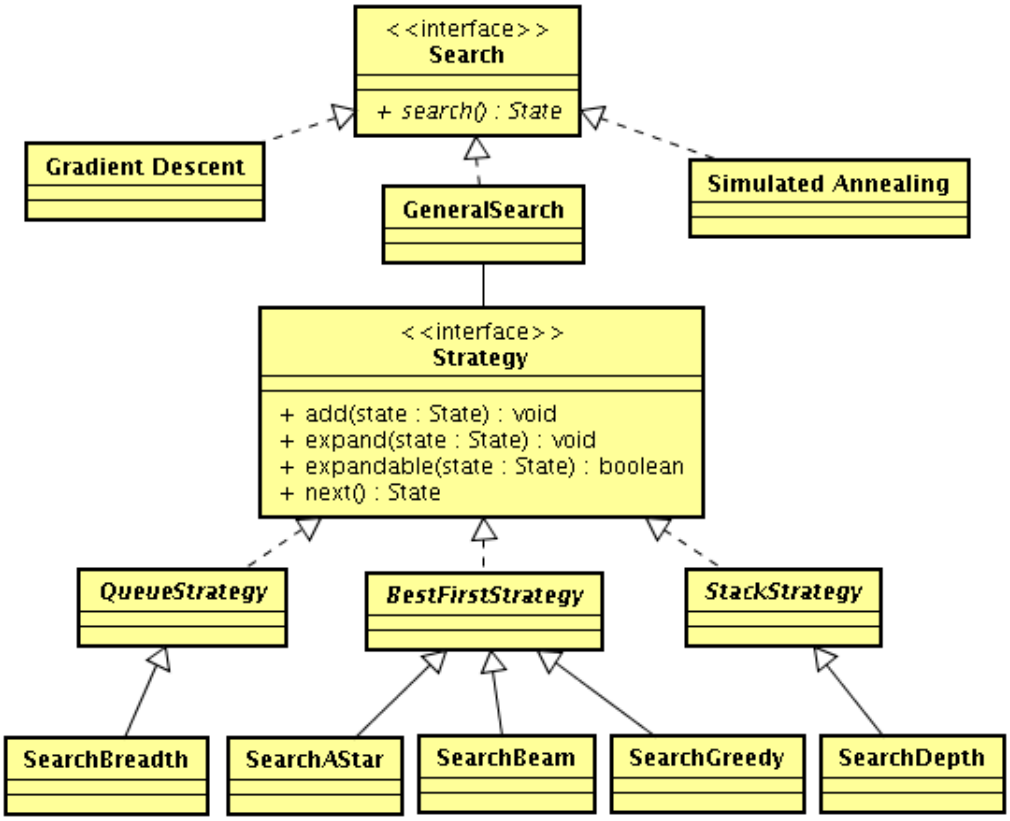

Figura 1: Diagrama de classes representando o modelo de buscas do simulador

1). O método *getY* retornar o valor da imagem do estado, já o método *produce* retorna um novo estado produzido a partir de dado x passado por parâmetro, necessitando apenas a o deslocamento ao valor do x. (Exemplo: Seja x o valor do estado atual, $x + 1$ é seu vizinho à direita, $x - 1$ é o à esquerda, a mesma regra vale para outros deslocamentos subsequêntes, $x + 2...x + n$).

A interface Strategy define os métodos que devem ser implementados para modelar um problema. Esta interface é requerida pelas implementações dos algoritmos de busca *GeneralSearch*, *DepthLimitedSearch*, *IterativeDeepeningSearch*. Três métodos são definidos por *Strategy*, $expand$, $expandable$ e $next$. O método $expand$, recebe como parâmetro um estado e tem como objetivo expandir os filhos deste estado, criando os novos nós da árvore de espaço de estados, mas nada é retornado. O método $expandable$, recebe como parâmetro um estado e retorna um valor booleano, verdadeiro caso o estado atual é expandível, falso, caso contrário. O método $next$ retorna o próximo estado a ser visitado.

5. Experimentos Realizados

Nos experimentos realizados os algoritmos implementados tinham a mesma tarefa, realizar uma seqüência movimentos no braço fazendo com que a mão deste braço no mínimo chegasse a um certo limiar perto no alvo (fixo ou em movimento), um objeto circular. Em uma das tarefas o alvo foi coloado em movimento e foi avalido como cada algoritmo se adaptava à nova situação, e se conseguir perseguir o alvo até encontrá-lo no espaço 3D.

O computador onde foi realizado estes testes foi um *Intel Pentium 4 2.4GHz* com 512MB de memória *RAM*, utilizando a máquina virtual *Java* versão *1.6.0-b105*, e um sistema operacional *GNU/Linux*. A versão do *Java 3D* foi a 1.5.0.

5.1. Teste com alvo fixo

Nestes testes o alvo foi posicionado em um local fixo no espaço 3D, dentro do limite de alcançe do braço. A seguir, mostraremos os resultados divido por cada classe de buscas, primeiramente as exaustivas, após as heurísticas e por fim as locais.

5.2. Buscas Exaustivas

As buscas exaustivas não utilizam nenhum recursos que auxiliam na busca ao alvo, desta forma, nenhum algoritmo conseguir encontrar o alvo nas circustâncias propostas. Os resultados são mostrados na tabela abaixo.

Tabela 1: Resultados com as buscas exaustivas

Algoritmo	Limite	Solução?	Tempo (ms)
Busca em Largura	-	Loop	-
Busca em Profundidade Limitada	300	Não	-
Iterative Deepening	10	Memória insuficiente	-

5.3. Buscas Heurísticas

As buscas heurísticas demostraram boa eficiência neste teste. Todos os algoritmos encontraram a meta. Os resultados são mostrados na tabela abaixo.

Tabela 2: Resultados com as buscas heurísticas

Algoritmo	Profundidade	Passos	Tempo (ms)
A*	6	18	305
Greedy	11	18	277
Beam (beam-width = 2)	11	12	202
Beam (beam-width = 5)	11	15	369
Beam (beam-width = 8)	11	18	303

5.4. Buscas Locais

As buscas locais tentam otimizar um resultado, desta forma, elas têm um maior número de passos, porém se mostraram mais rápidas e com menor consumo de memória. Todos os algoritmos encontraram o resultado. Com o algoritmo Simulated Annealing foi utilizada uma função de temperatura $T = T - 1$ com $T = 200$ de temperatura inicial. Com o algoritmo Gradient Descent foi limitada para 100 iterações. Na tabela abaixo os resultados.

Tabela 3: Resultados com as buscas locais

Algoritmo	Passos	Tempo (ms)
Simulated Annealing	200	74
Gradient Descent	100	45
Gradient Steepest Descent	100	42

Para testar o algoritmo *Simulated Annealing*, foi implementado um modelo de estado (extendendo a classe abstrata *SimulatedAnnealingState*), com uma função arbitrária (1) para obter a imagem (y) do estado x.

$$f(x) = ((x - 50)^3/100 + 1000 * cos(x)) \quad (1)$$

Tabela 4: Resultados dos experimentos com o problema dos vasilhames

Algoritmo	Limite	Solução?	Tempo (ms)
Busca em Largura	-	Sim	2
Busca em Largura Limitada	9	Sim	2
Busca em Largura Limitada	5	Não	4
Busca em Largura com aprofundamento iterativo	15	Sim	145
Busca em Profundidade Limitada	7000	Não	524

O estado inicial escolhido foi o 20, com temperatura reduzindo linearmente (decrementando de 1) com valor inicial em 20. Em 7 milisegundos o *Simulated Annealing* encontrou o estado $(x, y) = (3; -2028, 22)$, como o melhor. Na **Figura 2** podem ser vistas as iterações, a iteração 18 é o melhor mínimo local encontrado, podemos visualisar que o *Simulated Annealing* consegue, através da sua possibilidade de passo ruim, escapar de mínimos locais.

6. Considerações Finais

A Inteligência Artificial fornece ferramentas e técnicas que auxiliam na resolução de vários problemas reais. Os algoritmos aqui apresentados resolvendo problemas relativamente simples são usados em diversos problemas complexos, como vimos nos trabalhos correlatos. Os mecanismos da Inteligência Artificial podem ser utilizados em várias áreas da tecnologia da computação.

Neste trabalho podemos observar o poder e a sofisticação das técnicas, pois é necessário um pequeno ajuste na modelagem dos problemas, para que os algoritmos consigam solucionar os problemas.

Referências

Carson, Y. and Maria, A. (1997). Simulation optimization: Methods and applications. In *Winter Simulation Conference*, pages 118–126.

Ferguson, D., Likhachev, M., and Stentz, A. T. (2005). A guide to heuristic-based path planning. In *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)*.

Forsyth, D. A. and Ponce, J. (2002). *Computer Vision: A Modern Approach*. Prentice Hall.

Funge, J. D. (2004). *Artificial Intelligence For Computer Games: An Introduction*. A. K. Peters, Ltd., Natick, MA, USA.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1972). Correction to "a formal basis for the heuristic determination of minimum cost paths". *SIGART Bull.*, (37):28–29.

Korf, R. E. (1995). Space-efficient search algorithms. *ACM Comput. Surv.*, 27(3):337–339.

Russell, S. and Norvig, P. (1995). *Artificial intelligence: A modern approach*.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

Vaziri, K., Nozick, L. K., and Turnquist, M. A. (2005). Resource allocation and planning for program management. In *WSC '05: Proceedings of the 37th conference on Winter simulation*, pages 2127–2135. Winter Simulation Conference.

Xi, B., Liu, Z., Raghavachari, M., Xia, C. H., and Zhang, L. (2004). A smart hill-climbing algorithm for application server configuration. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 287–296, New York, NY, USA. ACM Press.

Yun, I. and Park, B. B. (2006). Application of stochastic optimization method for an urban corridor. In *WSC '06: Proceedings of the 37th conference on Winter simulation*, pages 1493–1499. Winter Simulation Conference.