VINICIUS RODRIGUES DE MORAES

SIMULATED ANNEALING APLICADO AO PLANEJAMENTO DE CAMINHOS PARA ROBÔS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia.

São Paulo
2003

VINICIUS RODRIGUES DE MORAES

SIMULATED ANNEALING APLICADO AO PLANEJAMENTO DE CAMINHOS PARA ROBÔS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia.

Área de Concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Marcos de Sales Guerra Tsuzuki

São Paulo
2003

"All we have to decide is what to do with the time that is given us." – Gandalf, the Grey ("The Lord of the Rings", J.R.R. Tolkien).

Este trabalho é dedicado com muito Amor às Mulheres de minha vida, Cristinas, e a Ricardo Rodrigues de Moraes pai e filho, dois grandes Homens.

AGRADECIMENTOS

O autor foi parcialmente patrocinado pela CAPES, tendo bolsa por aproximadamente metade deste Trabalho.

Muito grande a minha gratidão ao Professor Doutor Marcos Tsuzuki, meu orientador neste trabalho.

Obrigado ao Professor Sotelo, meu orientador no Projeto de Formatura, trabalho que colaborou na confecção deste.

Agradecimentos também seguem a minha família, à Cristina e meus amigos (Macel, Fred, Jeff, Pedrão, Dodô e Thiago deram enorme apoio técnico). Todos eles me apoiaram muito, seja com conhecimento técnico, compreensão e inestimável apoio moral.

Novamente, sou muito obrigado a Gary Gygax, Dave Arneson, Hal Jordan, Richard Garriot e J.R.R. Tolkien, que mantêm o sonho vivo.

RESUMO

O *Simulated annealing* é um algoritmo de otimização baseado na exploração Monte Carlo de um espaço n-dimensional. Podemos citar duas características importantes dos algoritmos: a primeira é que ele procura em um espaço contínuo com um custo que não cresce com a complexidade do espaço considerado; a segunda é que ele pode evitar mínimos locais não-globais, gerando soluções que são pelo menos quase-ótimas. Assim sendo, ele é indicado para o planejamento de trajetórias em ambientes cuja decomposição é computacionalmente complexa, pois trabalha diretamente sobre o espaço contínuo. Como o *Simulated annealing* não tem problemas em lidar com tais ambientes, o problema de encontrar uma trajetória ótima se transforma em um problema de otimização de uma função que estima o custo total da trajetória. Assim sendo, a proposta deste trabalho é utilizar o *Simulated annealing* – melhorado por um conjunto de heurísticas – para planejar trajetórias poligonais com baixo custo em espaços contínuos. Faz-se comparações de desempenho tanto entre várias implementações do *Simulated annealing* quanto entre ele e uma resolução de enfoque diverso: configuração espacial seguida de busca em grafos. Com tudo isto, avalia-se o efeito de cada regra heurística no aprimoramento do algoritmo *Simulated annealing* e quando a aplicação deste se justifica.

Palavras-chave --- Paramétrico; Planejamento de Caminhos; Planejamento de Trajetórias; Otimização; *Simulated annealing*; Recozimento Simulado; Simulador.

ABSTRACT

Simulated Annealing is an optimisation process based on the Monte Carlo exploration of an n-dimensional space. It has two characteristics of interest to this work: it explores continuous space with a cost that grows less than the complexity of the considered space; second, it is able to avoid non-global local minima, yielding results that are at least near optimal. It is thus suited for planning trajectories in environments whose decomposition is computationally complex (for it does not need one at all, as it works over the continuous space itself). As Simulated Annealing has no problem dealing with such environments, it can be applied to the problem of finding an optimal trajectory, by minimising the function that estimates the total cost of the trajectory. Our proposal, then, is to use Simulated Annealing - powered by a set of improvements - to plan low-cost, polygonal trajectories in continuous spaces. Performance comparisons are made, both between many Simulated Annealin

ERRATA

ABSTRACT

Simulated Annealing is an optimisation process based on the Monte Carlo exploration of an n-dimensional space. It has two characteristics of interest to this work: it explores continuous space with a cost that grows less than the complexity of the considered space; second, it is able to avoid non-global local minima, yielding results that are at least near optimal. It is thus suited for planning trajectories in environments whose decomposition is computationally complex (for it does not need one at all, as it works over the continuous space itself). As Simulated Annealing has no problem dealing with such environments, it can be applied to the problem of finding an optimal trajectory, by minimising the function that estimates the total cost of the trajectory. Our proposal, then, is to use Simulated Annealing - powered by a set of improvements - to plan low-cost, polygonal trajectories in continuous spaces. Performance comparisons are made, both between many Simulated Annealing implementations and between Simulated Annealing and a different approach: spatial configuration followed by graph searching. Then the effect of each heuristic rule on the performance of the Simulated Annealing algorithm is evaluated. Last, conclusions about when Simulated Annealing should be used are made.

Keywords --- Parametric; Path Planning; Trajectory Planning; Optimisation; Simulated Annealing; Simulator.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SA	Simulated Annealing
ASA	Adaptive Simulated Annealing
RFC	Reheat as Function of Cost
RG	Reaquecimento Geométrico

LISTA DE SÍMBOLOS

S	estado/posição/ponto inicial (start)
G	estado/posição/ponto final (goal)
n	n-ésimo nó; nó atual, genérico
g (n)	custo acumulado desde o nó S até o nó atual, n; é um número conhecido a cada instante
$\hat{h}(n)$	estimador heurístico de distância desde o nó atual, n, até o nó G
$\hat{f}(n)$	estimador do custo de S a G passando-se pelo nó n
k	constante de Boltzmann
T	parâmetro de temperatura, calculado pela função de recozimento
p	probabilidade de aceitação de solução de custo mais alto que a anterior
q	número de segmentos da trajetória procurada
T_o	temperatura inicial do sistema
lado	tamanho do lado do quadrado onde pode se encontrar a solução candidata na iteração seguinte
i	índice da temperatura
Ti	i-ésima temperatura
$T_{decremento}$	decremento da temperatura, i.e., o quanto se baixará a temperatura em cada oportunidade de resfriamento
α	fator de resfriamento (cooling factor)
L	dimensão do lado dos azulejos na decomposição do A*

1. INTRODUÇÃO

Quando um robô móvel está atravessando um ambiente, é importante que alcance seu alvo sem colidir com nenhum obstáculo e que também respeite as fronteiras do ambiente.

1.1 Planejamento de Trajetórias

O objetivo no planejamento de trajetórias de robôs (sem considerar sua dinâmica) é a geração de caminhos com mínimo custo e sem colisões entre as posições final e inicial dentro de um espaço de trabalho bidimensional W, o qual também contém objetos ou obstáculos fixos. O problema pode ser definido da seguinte forma (Latombe, 1993):

Sejam A, o robô, um corpo sólido simples movendo-se num espaço Euclidiano W, chamado ambiente ou espaço de trabalho, $\Re^n$, com $n=2$ ou 3. Sejam $B_1,\ldots,B_q$ corpos rígidos fixos em W, denominar-se-ão obstáculos. Assume-se que a geometria $A,B_1,\ldots,B_q$ e sua localização em W seja conhecida, e que não haja restrições físicas nos movimentos de A.

Em outras palavras, tendo-se uma posição inicial e uma posição final de A em W, uma trajetória τ deve ser obtida como uma seqüência de posições de A, tal que em cada uma destas posições ele deve evitar contato com qualquer B_i. A seqüência precisa começar com a posição inicial e terminar com a posição final. A trajetória precisa satisfazer $A \cap B_i = \varnothing$.

1.2 Revisão da Literatura

Há diversos estudos sobre planejamento de locomoção de robôs, usando os mais diversos enfoques. Yang e Meng (2000) utilizaram métodos globais para procurar possíveis caminhos no espaço de trabalho. Ong e Gilbert (1998) propuseram um modelo de planejamento de caminhos utilizando a distância do crescimento de penetração, que busca pelos caminhos que geram colisão ao invés de fazer esta procura no espaço livre, isto é, na parte desimpedida do ambiente.

Muitos modelos de redes neurais são propostos para o planejamento de locomoção do robô via aprendizado; por exemplo Muñiz *et al.* (1995) propuseram um modelo de rede neural para navegação dinâmica de um robô móvel que evita obstáculos pelo *reinforcement learning*. No entanto, o movimento planejado do robô, quando se usa enfoques baseados no aprendizado, não são ótimos, particularmente nas fases iniciais do aprendizado.

O *Simulated annealing* é um método de otimização introduzido por Kirkpatrick na década de 80 (Kirkpatrick *et al.*, 1983) e até hoje encontra larga aplicação, sendo ele adaptado e recebendo otimizações para casos específicos.

A busca em grafos e toda a análise comparativa de desempenho de seus algoritmos, com métodos de medição dos mesmos, encontra-se em Nilsson (1971; 1980).

1.3 Planejamento de trajetórias: contínuo *versus* discreto

Planejamento de trajetórias é um problema que possui diferentes enfoques de solução implementados. Muitos destes enfoques conseguem determinar soluções computacionalmente eficientes; por exemplo os métodos de busca em grafos, como a busca em largura, busca em profundidade, Dijkstra e sua evolução, o **A***. Entretanto, tais algoritmos baseiam-se na possibilidade de redução de espaço a um grafo com um número finito de nós e arcos. Essa representação do ambiente como um grafo finito é a etapa essencial conhecida como modelagem. Assim sendo, tais algoritmos baseados em grafos são adequados a problemas com um número finito de posições intermediárias de interesse (e um número finito de caminhos ligando tais posições). Um exemplo clássico é planejar um caminho num conjunto de ruas: apenas os cruzamentos e ligações entre eles (as próprias ruas) são de interesse ao problema. A aplicação prática de tal exemplo é o planejamento de trajetória de caminhões de entrega de uma empresa.

Porém, quando se trabalha em espaços contínuos, é necessário aplicar-lhes um pré-processamento para os mapear para um grafo. Isto visa a adequá-los aos algoritmos clássicos supracitados. Problemas de interesse em ambientes contínuos são: planejamento de trajetória de uma ferramenta no espaço e planejamento de caminho para um robô móvel em campo aberto (por exemplo, em busca de minas

terrestres ou na superfície lunar ou de Marte ou mesmo um chão de fábrica). Em todos estes casos há infinitas posições intermediárias relevantes e, portanto, infinitos caminhos que os conectam.

Tal pré-processamento também é chamado modelagem, digitalização, configuração espacial ou discretização. Ele associa regiões do espaço de trabalho (chamadas células) a nós do grafo e então usa informações da topologia do ambiente para determinar a conectividade e os custos de travessia.

A qualidade do planejamento depende da qualidade da configuração espacial: o número de células e suas formas podem causar um planejamento de custo muito alto, ou mesmo uma trajetória distante da ótima. Eventualmente, em ambientes de topologias complexas, achar uma configuração espacial satisfatória pode ser uma tarefa bem mais árdua que a própria busca pela trajetória no grafo.

Uma alternativa aos algoritmos que se utilizam de busca em grafos é trabalhar diretamente no espaço contínuo, com trajetórias definidas por um conjunto finito de parâmetros x, e tentar minimizar a função $C(x)$, definida como custo total da trajetória definida por x no espaço E. Dada a complexidade do espaço x, este problema se mostra particularmente adequado a algoritmos de otimização probabilística. Nossa proposta é enfocar o problema de planejamento de caminhos em espaços bidimensionais com trajetórias descritas por curvas poligonais e aplicar o método de otimização conhecido como *Simulated annealing* a tais trajetórias, achando, assim, soluções ótimas ou ao menos próximas daquelas, isto é, soluções quase-ótimas (*near-optimum solutions*).

2. OTIMIZAÇÃO POR BUSCAS EM GRAFOS

A busca em grafos será agora introduzida, pois é ela que se dá após a decomposição espacial, no método que se comparará com o *Simulated annealing*.

Após o estudo da busca em grafos em si, veremos como ela é utilizada na busca por caminhos de baixo custo. Exemplos serão expostos e resolvidos por ambos os métodos (*Simulated annealing* e decomposição espacial seguida de busca em grafos), para estudos de casos. Após os exemplos, seguem conclusões sobre quando se utilizar cada método.

2.1 Estados e Operadores

Usar-se-á um exemplo para introduzir a busca em grafos: "Jogo dos 15".

Estado é o conjunto de variáveis. Ao conjunto de possíveis estados se dá o nome de Espaço de Estados.

Para discutir métodos de solução de problemas deste tipo é necessário introduzir as noções de estados e operadores. No jogo dos 15, cada estado do problema é uma particular configuração dos quadradinhos, como se pode ver na Figura 1.

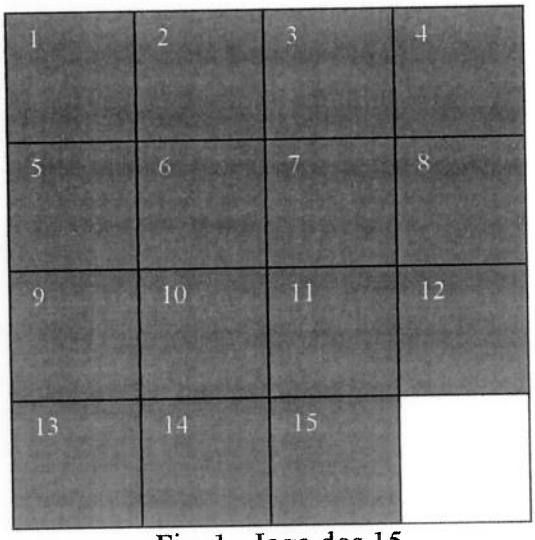

Fig. 1 . Jogo dos 15.

Tem-se dois estados como estados especiais: o estado-início e o estado-objetivo.

O espaço de estados alcançável a partir do estado inicial consiste em todas as configurações legais que podem ser obtidas movendo-se os quadradinhos da maneira permitida pela regra. Este número de estados pode ser muito grande ou mesmo infinito. Ainda no caso Jogo dos 15, este número é grande: para uma posição inicial qualquer, vale 0,5*16!. Isto porque o total de posições possíveis no quebra-cabeças é 16! e porque há dois grafos (espaços de estados) distintos, disjuntos: um "par" e outro, "ímpar", cada qual tendo 0,5*16! posições. Dada uma posição qualquer par do grafo, é impossível se chegar a uma posição ímpar e vice-versa. Para se mudar de uma posição par para uma ímpar (e vice-versa), entretanto, é bem fácil: troca-se de lugar uma peça do quebra-cabeças com a sua vizinha. É claro que isto se dá no mundo real; no espaço de estados, não há operador "troque duas peças entre si".

A modelagem por "espaço de estados" é muito empregada nas áreas de Controle e de Pesquisa Operacional.

Um operador transforma um estado em outro (Nilsson, 1971; Nilsson, 1980); no exemplo em questão, há quatro operadores: "mover branco para a direita" (MD), "mover branco para a esquerda" (ME), "mover branco para cima" (MC) e "mover branco para baixo" (MB).

Na linguagem de espaço de estados, uma solução para um dado problema será qualquer seqüência de estados que transformar um estado-início em um estado-objetivo. Isto se faz construindo-se seqüências de operadores (adicionando um operador por vez) até que o objetivo seja alcançado.

É útil imaginar o espaço de estados alcançável a partir de um dado estado inicial como um grafo. Neste grafo, os nós representam os estados (o nó-raiz é o estado-início) e os arcos orientados que os conectam são os operadores.

2.2 Reduzindo Problemas a Subproblemas

A noção de subproblemas será agora introduzida, pois é utilizada no algoritmo-base do **A***. A idéia básica é criar um conjunto de subproblemas, de modo que a solução de cada um destes implique na solução do problema original (Nilsson, 1971; Nilsson, 1980).

Considere o problema "Ir do Brasil para os EUA" (Moraes, 1999). Ele pode ser decomposto em "Ir do Brasil para o Panamá" e "Ir do Panamá até os EUA" (sempre

via terrestre, lembrando que o Panamá é uma constrição). É algo semelhante a isto que será utilizado no **A***: achar, no espaço de estados, o caminho que minimize o custo de transformação do estado-início ("Brasil") no estado-objetivo ("EUA") passando por um dado ponto ("Panamá"). Tal problema se reduz a minimizar, simultaneamente, o custo do início até dado ponto e deste ponto até o fim.

Cada um dos subproblemas pode ser olhado como um novo problema, que pode ser resolvido, bem como o problema original, via espaço de estados ou criação de novos subproblemas e assim por diante. A idéia é descer tanto no nível de complexidade de um problema que se chegue em novos problemas que sejam primitivos, isto é, cujas soluções sejam triviais. Esta é a idéia do chamado "Método da Redução do Problema".

2.3 Elementos da Solução de Problemas por Busca em Grafos: Representação e Busca

Qualquer que seja o método escolhido para se solucionar um problema, uma busca pela solução é requerida.

Entretanto, antes da busca em si, deve haver uma grande preocupação com a modelagem, que é a representação do sistema. Isto é condição necessária e não-suficiente para um bom desempenho global do sistema solucionador de problemas.

2.4 Descrições de Estado

Quando se faz a formulação via espaço de estados de um problema, é preciso não confundir o que se representa (estado) com como se representa - as descrições de estado (Nilsson, 1971; Nilsson, 1980).

No exemplo do Jogo dos 15, um estado é cada possível configuração das peças. Sua descrição, entretanto, pode ser feita por qualquer uma das seguintes estruturas de dados: matrizes, árvores, vetores, listas ligadas, *heaps*, *strings* simbólicos entre outros.

Algo que deve ser mantido em vista é a utilização de uma forma de representação que facilite a aplicação dos operadores, acelerando, assim, a transformação de um estado em outro. Pensando nisto e em facilidade de

programação, utilizou-se, na implementação do programa gerado (o **AStar**), uma representação matricial.

Assim sendo, pode-se entender os operadores como funções parciais que, aplicadas ao seu domínio (o conjunto das descrições de estados possíveis), têm como valor uma descrição. São, assim, do tipo:

$$f : D \Rightarrow D$$

Em que D é o conjunto das descrições dos estados possíveis.

Utilizou-se o conceito de funções parciais porque elas não são funções, dado que nem sempre um operador pode ser aplicado a todos os estados: por exemplo, na Figura 1, MD e MB não se aplicam.

É bom lembrar que não basta achar uma solução, isto é, um caminho qualquer até o objetivo: é necessário achar o "melhor" caminho (Moraes, 1999). Por "melhor" entende-se o caminho que satisfaça um critério de avaliação pré-definido (Nilsson, 1971; Nilsson, 1980). Neste caso, o critério é minimizar a soma dos custos dos operadores. Após tal solução, pode-se afirmar que se tem uma solução admissível.

Com o que se viu nesta seção, pode-se resumir as etapas de representação de um problema via espaço de estados: duas coisas precisam ser especificadas cuidadosamente:

1) a forma da descrição de estados;

2) o conjunto de operadores e seus efeitos nas descrições de estados.

2.5 Notação de Grafos

É interessante expressar o espaço de estados como um grafo orientado. Tal representação é particularmente útil para se analisar e comparar os vários métodos de busca em um espaço de estados. Introduzir-se-á, portanto, um pouco da terminologia de grafos.

Um grafo é constituído por um conjunto de nós, finito ou não. Alguns pares de nós são conectados por arcos, que são orientados, i.e., apontam de um dos nós do par ao outro: daí as denominações arco orientado e grafo orientado. O grafo de onde "vem" o arco orientado é chamado de pai (ou predecessor) do outro nó, que é chamado de filho (ou sucessor) do primeiro. Fontes são nós sem predecessores; sumidouros são nós sem sucessores.

Se existe uma trilha de um nó qualquer *u* para outro *v*, *v* é dito acessível a partir de *u*, ou descendente de *u*. O nó *u* é, assim, ancestral de *v*.

No caso de espaço de estados, os nós representam as descrições dos estados, e os arcos orientados representam os operadores. No caso do robô em seu espaço de trabalho, são, respectivamente, as posições do robô e seus movimentos (de uma posição a outra) (Moraes, 1999).

Faz-se também necessário associar um peso a cada arco: tal valor representa o custo de se aplicar o operador correspondente. No ambiente de trabalho do robô, isto representa o custo de viagem (*travelling cost*) de uma célula a outra, também chamado de custo de travessia, de modo que os arcos são associados aos caminhos (e os pesos dos arcos são os custos dos caminhos). Será usada a notação $c(n_u,n_v)$ para denotar o custo do arco que liga o nó *u* ao nó *v*, i.e., para se "ir" do ponto representado pelo primeiro nó para o segundo (ou ainda para se "levar" o sistema do estado *u* para o *v*). O custo de uma trilha qualquer entre dois nós será a soma dos custos de todos os arcos que ligam os nós de tal trilha.

Percebe-se, portanto, que o problema de achar uma seqüência de operadores que transforme um estado em outro é equivalente ao de achar uma trilha num grafo (Nilsson, 1971; Nilsson, 1980). Perceba que se chama o caminho no grafo de trilha para se diferenciar do caminho do robô no mundo real, i.e., a trajetória a ser achada.

A diferença entre trilha e caminho (trajetória) pode ser claramente compreendida comparando-se as Figuras 2 e 3 (Moraes, 1999). A Figura 2 é relacionada à estrutura de dados: o grafo orientado mostra as conexões existentes entre os estados (nós) 1, 2, 3, 4 e 5. Não somente isto, mostra também o custo para se ir de um nó a outro. A Figura 3, por sua vez, mostra o mundo real, o ambiente de trabalho em que o robô irá se locomover.

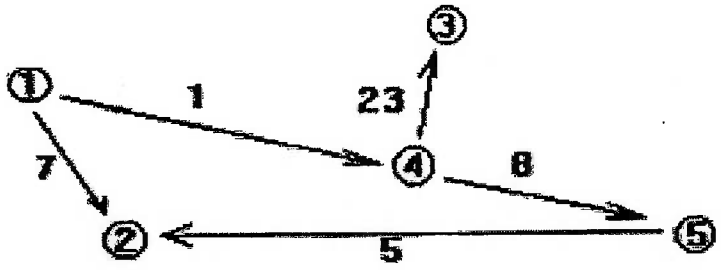

Fig. 2 . Exemplo de grafo orientado.

Apresentada a nomenclatura, vê-se agora na Figura 2:

- Nós: 1,2,3,4 e 5.
- Trilhas: 1→2, 1→4→5→2, 1→4→3 e suas sub-trilhas (1→4, 4→5→2 etc).
- Arcos: 1→2, 1→4,4→5,5→2 e 4→3.
- Pais ou predecessores de 1: não há: 1 é fonte.
- Pais ou Predecessores de 2: 1 e 5.
- Sucessores de 4: 3 e 5.
- Sucessores de 2: não há: 2 é sumidouro.
- O peso do arco 4→3 é 23. O peso do arco 4→5 é 8.

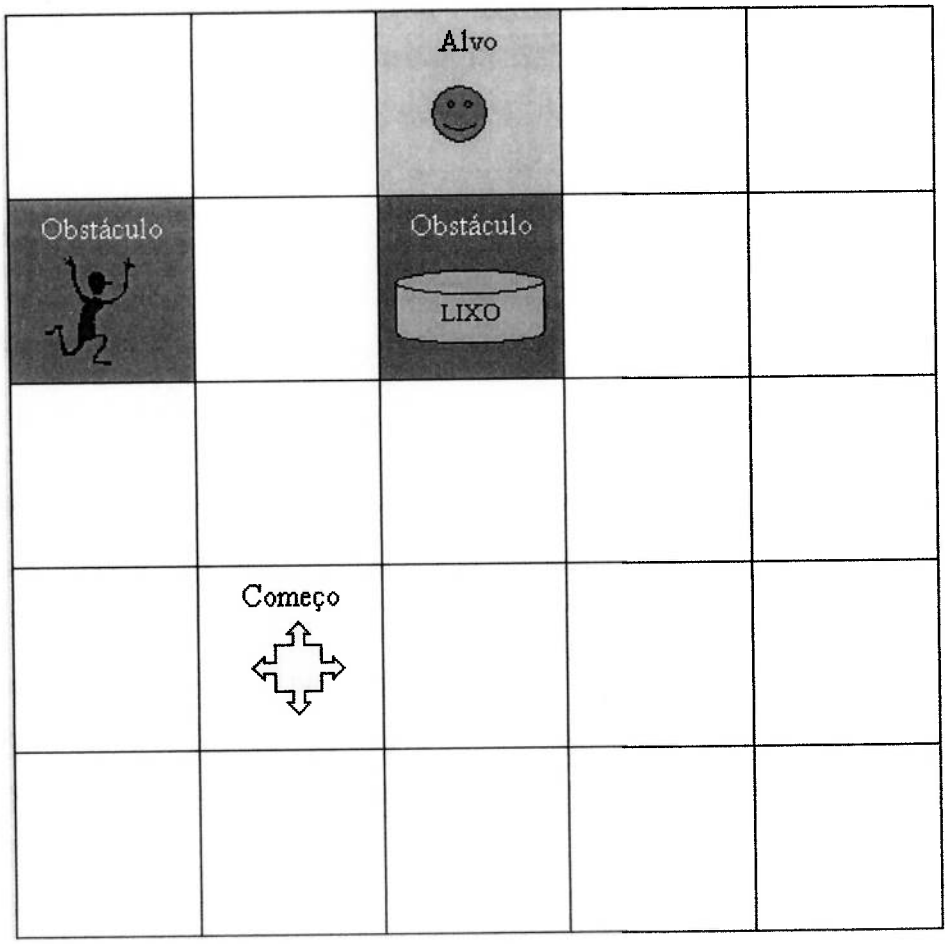

Fig. 3 . Exemplo do ambiente e de sua discretização.

2.6 Otimização no Espaço de Estados.

Com relação a espaço de estados, o que é otimizar? É tentar achar um conjunto de operadores com duas propriedades: que levem o sistema do estado-início até o estado-objetivo; que minimizem algum critério pré-definido (Nilsson, 1971; Nilsson, 1980). No grafo, isto é equivalente a achar uma trilha que minimize o custo entre os dois nós correspondentes aos dois estados, os quais se denominará *S* e *G* (*start* e *goal*), respectivamente.

Um grafo pode ser especificado explicitamente ou implicitamente (Nilsson, 1971; Nilsson, 1980); o primeiro modo foi utilizado no trabalho (Moraes, 1999) que resultou no programa **AStar**, a ser analisado. Um grafo especificado explicitamente é dito um grafo explícito. Um grafo especificado implicitamente é dito um grafo implícito.

Diz-se que a especificação é explícita quando os nós, arcos e custos destes últimos são dados explicitamente. Isto pode ser implementado, por exemplo, através de uma tabela que liste todos os nós do grafo, seus sucessores e os custos associados aos arcos que levam até tais sucessores. Para um número muito grande de arcos, percebe-se facilmente, que uma especificação explícita é impraticável; e para um número infinito de arcos a especificação é impossível.

É interessante notar que a representação utilizada incluiu, além do que se citou acima (que já é suficiente para descrever o grafo), outra informação na descrição de cada nó: o nó-sucessor. Tal dado é importante quando se "volta" no grafo após se ter encontrado o nó-objetivo. Essa volta é exatamente a construção da resposta, a trilha de custo mínimo.

Aparece, assim, a especificação implícita, em que um conjunto de nós é dado como estado inicial. Neste trabalho, somente o caso com um estado inicial e um estado-objetivo é de interesse. Tem-se um operador-sucessor γ tal que pode ser aplicado a qualquer nó, gerando sucessivamente todos os seus sucessores e os custos dos arcos associados. Se tal nó inicial for o nó-raiz, descreve-se, com tal procedimento de sucessivas aplicações, o grafo todo. Perceba que isto é feito com uma representação bem mais econômica, i.e., somente em termos do conjunto de estados inicinicior γ.

O processo de procurar num espaço de estados uma seqüência de operadores que seja solução (i.e., que leve ao estado-objetivo) corresponde a fazer explícita uma porção de um grafo implícito, porção esta suficiente para incluir um nó-objetivo (Nilsson, 1971; Nilsson, 1980). Procurar em grafos desta maneira é, assim, um elemento central da solução de problemas via espaço de estados.

Com tal exposição (sobre as etapas básicas de uma busca), conclui-se que se deve buscar qualidade e desempenho para todas as etapas: negligenciar qualquer uma delas pode causar impacto no resultado final.

A Discretização

Quando se vai fazer utilização de algoritmos deste tipo, deve-se escolher a unidade básica de discretização. Geralmente utiliza-se polígonos simples, existindo assim a representação por quadrados (azulejos, *tiles*) e por "hex" (hexágonos) (Moraes, 1999). Triângulos também poderiam ser utilizados, mas a alcançabilidade ao espaço próximo é mais facilmente implementada pelos dois primeiros modos: o quadrado, quando se usa - além dos eixos horizontal e vertical – os eixos diagonais, liga-se a outros oito quadrados. No caso dos "hex", são seis hexágonos adjacentes, um em cada lado do hexágono que se está analisando. Se fosse um triângulo, haveria três triângulos adjacentes e outros nove próximos, com dois problemas: haveria três tipos de ângulos e a análise passaria exatamente por cima dos triângulos adjacentes.

Também se deve escolher a respeito da possibilidade de movimento diagonal para o caso de se escolher o quadrado como elemento básico da configuração. Perceba que, no caso da discretização em hexágonos, os movimentos possíveis são aqueles nas seis direções intuitivas (para os seis hexágonos adjacentes).

Estas duas escolhas são ligadas ente si e devem ser feitas na etapa de concepção. Assim, desde tal etapa se decidiu por representação e discretização por azulejos quadrados e pela permissão de movimento diagonal. Isto torna a árvore de busca uma *octree*.

Ótimo x Admissível: Confusão Comum

É muito interessante ressaltar a diferença entre estes dois importantes conceitos (Nilsson, 1971; Nilsson, 1980):

* Admissível: é a garantia de mínimo custo entre dois estados num dado espaço de estados, para dada heurística. Custo mínimo de execução. É característica da trajetória. Sinônimo, neste trabalho, de eficaz.

* Ótimo: para se achar uma trilha qualquer (admissível ou não) num grafo, garante que se expandem o menor número possível de nós na busca. Custo mínimo de planejamento. É característica do algoritmo de busca. Sinônimo, neste trabalho, de eficiente.

Assim, um dado caminho pode ser admissível ou não. E a busca para o achar pode ter sido ótima ou não. Isto faz de um processo como um todo (algoritmo de busca mais solução por ele encontrada): admissível ou não; ótimo ou não.

As quatro combinações entre estas características são possíveis. É claro que a combinação que interessa a este trabalho é "admissível e ótimo", ou seja, eficaz e eficiente, respectivamente (Moraes, 1999).

2.7 Algoritmos de Busca Consagrados

Segundo um critério referente à utilização da informação sobre o ambiente de trabalho, existem dois tipos básicos de buscas em grafos: busca cega e busca não-cega (Nilsson, 1971; Nilsson, 1980).

A busca cega é aquela em que não se leva em conta a posição do alvo (mesmo que se tenha tal informação). Exemplos de algoritmos "cegos":

- Força Bruta;
- Largura Primeiro (Busca em Largura);
- Profundidade Primeiro (Busca em Profundidade);
- IDDF (*Iterative-deepening Depth-first*).

No caso deste trabalho, tem-se a informação em questão (a posição do ponto-objetivo). Assim sendo, por que não a usar? Define-se, com tal motivação, a busca não-cega.

A busca não-cega é a que utiliza a informação desprezada pelos métodos cegos. Assim sendo, com a utilização do importante dado que é a posição-objetivo, obtém-se rápido direcionamento da busca em direção a ela. Exemplos de buscas que "enxergam":

- Melhor Primeiro;

- Dijkstra;
- **A***, que reúne as características positivas de Melhor Primeiro e de Dijkstra.

2.8 A Estrela dos Algoritmos de Busca: o algoritmo A*

O método implementado no programa **AStar** foi o **A***, exatamente por unir as características desejáveis dos métodos Melhor Primeiro e Dijkstra. A seguir compara-se o **A*** com os outros algoritmos tradicionais de busca em grafos. Depois, explica-se o **A***.

Comparações Entre o A e as Outras Buscas em Grafos*

Como introduzido acima, o **A*** é, na pior das hipóteses, igual aos dois algoritmos que o originaram - Dijkstra e Melhor Primeiro - (Moraes, 1999), e aqueles podem ser vistos como casos específicos deste. Assim, já se percebe as vantagens, ainda sem argumentação em mais baixo nível, do **A*** em relação aos demais métodos de busca. Ainda assim, a argumentação em termos mais técnicos se faz necessária.

O algoritmo Melhor Primeiro não é admissível, mas muito eficiente na busca. O algoritmo desenvolvido por Dijkstra é admissível mas não é ótimo, ou seja, leva um tempo maior do que se desejaria para achar o melhor caminho. Isto posto, por que não unir as características desejáveis de ambos? Nasce, assim, o **A***. Algo muito semelhante a esta discussão será feito com o *Simulated annealing*: serão usados um processo somente eficiente, um somente eficaz e finalmente um outro que une estas duas características desejáveis.

É interessante salientar que o Melhor Primeiro é muito interessante quando não se impõe admissibilidade, i.e., quando se faz necessário achar algum caminho até o alvo com o mínimo tempo de busca possível. Neste caso, o Melhor Primeiro é mais adequado que o **A***. Também neste caso se faz analogia com o *Simulated annealing*: pode-se usar o **SA** que é somente eficiente para gerar rapidamente algum caminho até o objetivo.

Para o desenvolvimento tecnológico atual, o tempo de cálculo de um caminho via busca em grafos é ordens de grandeza menor que sua execução (leia-se o deslocamento em si). Assim sendo, o **A*** é bem mais interessante que o Melhor Primeiro, pois o pequeno tempo adicional perdido em busca com relação àquele gera

uma enorme economia de tempo, espaço e risco, quando da execução da trajetória. Para o caso em que o processador fosse muito lento ou que calculasse caminhos para muitos robôs ao mesmo tempo, ter-se-ia um caso em que o custo de busca não mais seria desprezível, e assim o Melhor Primeiro cresceria em importância.

É um enfoque interessante pensar que o Melhor Primeiro só consegue economizar tempo (de cálculo), enquanto o **A*** minimiza o que quer que seja usado como peso dos arcos. No caso deste estudo, o interesse é pôr o custo de travessia como peso, pois é o custo total de *S* a *G* que se quer minimizar.

O algoritmo Dijkstra acha o mesmo caminho que o **A***, porém com um custo computacional maior. O fato de o **A*** ser admissível e ótimo quer dizer que nenhum outro algoritmo de busca, utilizando a mesma heurística e a mesma discretização, acha um caminho admissível expandindo menos nós que o **A*** (isto equivale a dizer que ele tem mínima arborescência) (Nilsson, 1971; Nilsson, 1980).

Assim, a conclusão a que se chega é: o algoritmo de busca em grafos que mais interessa ao problema deste trabalho é o **A***. Isto porque, sendo ele ótimo e admissível, o seu conjunto planejamento/execução é o melhor possível (dentre os algoritmos de busca em grafos) para os problemas que este trabalho se propõe a resolver. É importante que se tenha em mente que se está buscando caminhos de baixo custo, entre dois pontos dados, para um único robô, em um ambiente bidimensional, com mínima computação possível. Assim sendo, um caminho "qualquer" não vale, deve ser o de mínimo custo, e achado com o mínimo de esforço computacional. Por isto é que se insiste em qualidade e desempenho (ou, respectivamente, admissibilidade e que seja ótimo). É importante lembrar que o Melhor Primeiro é admissível, mas não ótimo (pode demorar mais que o **A*** para gerar resposta ótima).

*Funcionamento do A**

Antes de explicar o algoritmo em si é necessário introduzir dois conceitos que serão utilizados com freqüência (Moraes, 1999).

Expandir um nó é verificar os operadores que se pode aplicar a cada nó (isto é, aplicar-se os operadores possíveis dentre dos oito existentes: N, NE, EE, E, S, SW, W e NW), obtendo-se assim os filhos daquele nó. Perceba que cada um dos operadores corresponde à movimentação do robô para um dos oito azulejos

adjacentes ao azulejo atual, localizados nos sentidos cardeais e colaterais em relação a este.

Nó mais promissor que outro é um nó que obteve uma melhor nota no sistema de medida imposto. Tal sistema é exatamente o conjunto de regras heurísticas, as funções de avaliação, que são o cerne de boa parte dos algoritmos daquilo que se convenciona chamar de Inteligência Artificial. É também uma parte bastante crítica e que requer um grande tempo de ajuste fino de parâmetros, estimadores e avaliadores. É, assim, o modo de se comparar as possíveis soluções, decidindo-se por esta ou aquela para se convergir para a resposta do problema.

O algoritmo

O algoritmo **A*** se baseia na minimização - para dada heurística e dado método de configuração - da função de avaliação $\hat{f}(n)$ (Moraes, 1999), dada pela equação

$$\hat{f}(n) = g(n) + \hat{h}(n) \qquad (1)$$

em que:

→ n é o nó atual, genérico;

→ g *(n)* é o custo acumulado desde o nó S ("S" de *start*), inicial, até o nó atual, n; é um número conhecido a cada instante (e não um estimador);

→ $\hat{h}(n)$ é o estimador heurístico de distância desde o nó atual, n, até o nó G ("G" de *goal*); o índice ^ (acento circunflexo) indica ser um estimador (e não um número conhecido);

→ $\hat{f}(n)$, sendo a soma de um número com um estimador, é um estimador. Das definições de g *(n)* e de $\hat{h}(n)$, tem-se que $\hat{f}(n)$ é o estimador do custo de S a G passando-se pelo nó n.

É interessante rever o exemplo da ida do Brasil até os Estados Unidos passando pelo Panamá. Considerando-se uma iteração (a iteração em que a busca está passando pelo Panamá): o Brasil seria o ponto (nó) S, o Panamá seria o nó n e os EUA, o G.

Quando se chega ao nó-solução, G, tem-se $\hat{h}(n) = 0$ e, é claro, nada mais a se estimar, então todo o valor de $\hat{f}(n)$ é composto por $g(n)$ e, assim, o custo é real, número conhecido.

Agora que já se entende o funcionamento do algoritmo **A***, abaixo apresenta-se o mesmo em português estruturado (Nilsson, 1971; Nilsson, 1980):

(l) Põe o nó-início S numa lista chamada *ABERTOS* e calcula $\hat{f}(S)$

(2) Se lista *ABERTOS* está vazia, algoritmo termina, dando erro em encontrar nó-solução; i.e., não há trilha entre os dois nós; (se não está vazia, continua)

(3) Remove da lista *ABERTOS* o nó com o menor valor de $\hat{f}(n)$ e o põe numa lista chamada *FECHADOS*. Chama este nó de n. Se houver dois nós com mesmo valor de $\hat{f}(n)$, desempata mínimo $\hat{f}(n)$ arbitrariamente, mas sempre em favor de solução (i.e., se há um nó qualquer e um nó-solução, escolhe-se este)

(4) Se n é nó-solução, o algoritmo termina dando o caminho-solução, obtido rastreando-se de volta os apontadores (postos no passo 5); (se n não é nó-solução, continua)

(5) Expande o nó n, gerando todos seus sucessores. Se não há sucessores, vai para (2). Para cada sucessor n_i, calcula $\hat{f}(n_i)$. Põe estes sucessores na lista *ABERTOS*, associando-os aos valores recém-calculados de $\hat{f}(n)$, e lhes dá apontadores de volta para n, seu nó-pai

(6) Vai para (2).

Assim, o que se faz é: analisam-se, em cada nó do grafo (posição do ambiente, no mundo real), os nós gerados pelos operadores possíveis para aquele nó. Ordenam-se estes nós-filhos segundo o seguinte critério: os nós mais promissores são postos numa lista chamada *ABERTOS* (pois são os nós ainda abertos a análise, ainda podem ser parte da solução). Eles são colocados na lista em primeiro lugar, de modo FIFO (*first-in-first-out*, uma fila). São estes os nós que, no passo seguinte, serão expandidos: seus nós-filhos serão analisados, e assim por diante, com seus sucessores, passo após passo.

*A Heurística e o A**

A definição de heurística será feita pelo exemplo de jogadores de xadrez, sejam eles humanos ou computadores. Depois, será exposta a heurística específica do **A***.

Heurística é uma regra ou conjunto de regras que se usa, vinda de conhecimento adquirido em observações anteriores, possivelmente guardados em bancos de dados (Moraes, 1999). No xadrez, por exemplo, atribui-se a cada jogada valores às peças e pesos para tais valores. Tais pesos e valores são baseados nas qualidades das peças, em suas posições e no momento do jogo. A primeira regra se dá porque uma rainha vale mais que um bispo e também vale mais que uma torre, pois é "uma soma dos dois" em termos de possibilidades de movimentos. A segunda regra vem do fato que uma peça no meio do tabuleiro vale mais que uma em um de seus quatro lados (e nos lados vale mais que nos quatro vértices), pois alcança mais casas, potencialmente, resultando em domínio de centro, que em muitas vezes decide o jogo. A terceira regra advém do conhecimento de que um cavalo, por exemplo, é mais valioso no começo do jogo do que no meio ou fim (isto por poder "pular" peças). Sublinhou-se o termo "conhecimento" porque a heurística é exatamente isto: regras não-determinísticas que se observou e aprendeu. Isto para que tais dados ajudem a estimar casos futuros. A aplicação de tal tipo de conhecimento é parte importante em tomadas de decisão.

Este método baseia-se na garantia de que os nós mais promissores serão expandidos em primeiro lugar, o que direciona a busca ao ponto-objetivo. Isto ocorre até que se ache um nó-solução ou que se conclua que não há caminho entre os pontos S e G.

Este é, portanto, um método que não é de busca cega. O ganho de desempenho do **A*** (e das buscas não-cegas em geral) sobre os métodos de busca cega é considerável, e isto se dá exatamente por causa deste direcionamento da busca.

É exatamente a função heurística do **A*** – a função $\hat{h}(n)$ - a responsável por esta inteligência, a "intuição" do método (Moraes, 1999). É ela quem vai "perceber" quais nós são mais promissores. É exatamente nela que reside a diferença entre um bom e um mau **A***. Perceba ser tal intuição, dada por $\hat{h}(n)$, inexistente em Dijkstra (pois este somente se preocupa com o custo até o nó atual, i.e., $g(n)$). Perceba que, de modo inverso, $g(n)$ inexiste no Melhor Primeiro, algoritmo que somente computa e utiliza $\hat{h}(n)$, desprezando o custo até o nó atual. Melhor Primeiro, assim, somente

se preocupa com o custo do trecho ainda faltante até o alvo. Relembrando, o **A*** utiliza ambos os dados para buscar ser ótimo e admissível, combinando o desempenho de Melhor Primeiro e da eficácia de Dijkstra.

A conclusão é que a heurística, no caso do **A***, está justamente no $\hat{h}(n)$. Ele é o estimador.

No caso do **A***, tem-se que usar um $\hat{h}(n)$ que seja um sub-estimador do valor real (Nilsson, 1971; Nilsson, 1980), para que não se descartem nós válidos - e, assim, caminhos válidos. No caso de planejamento de trajetórias, utiliza-se um dos seguintes métodos estimadores:

1) distância *Manhattan*: estima-se a distância do ponto atual até o ponto-objetivo andando-se (e contando) somente por "ruas e avenidas de *Manhattan*", que são, respectivamente, nas direções leste-oeste e norte-sul; daí o nome. Isto quer dizer que somente se pode contar distância utilizando as direções cardeais. A fórmula que resume isto é:

$$\hat{h}(n) = dx(n) + dy(n) \qquad \textbf{(2)}$$

2) distância Euclidiana: estima-se a distância do ponto atual até o ponto-objetivo andando-se (e contando) por uma reta de distância mínima, a distância Euclidiana. Tal método é também chamado de "olhos de águia", pois uma águia pode ir do ponto atual ao ponto-objetivo pela reta de Euclides. A fórmula correspondente é:

$$\hat{h}(n) = \sqrt{dx(n)^2 + dy(n)^2} \qquad \textbf{(3)}$$

3) atalho diagonal: estima-se a distância do ponto atual até o ponto-objetivo andando-se (e contando) numa das diagonais. Faz-se isto até se estar na mesma linha (ou coluna) que o ponto-objetivo. A partir disso, vai-se "reto" até ele, ou seja, numa das quatro direções cardeais. A fórmula é mais complicada, e varia com o quadrante, tendo algoritmo semelhante àqueles de traçar retas;

4) *max (dx, dy)*: o nome do estimador, que já é a própria fórmula, já dá uma idéia de que este método estimador é até certa parte igual ao *Manhattan*. Assim sendo, conta-se dx e dy do mesmo modo (somente nas direções cardeais), mas eles não são somados, mas sim comparados. De tal

comparação, conclui-se e se utiliza o maior entre os dois.

O segundo e o terceiro métodos estimadores, acima mostrados, são bastante interessantes no caso de se permitir movimento diagonal, que é o caso deste trabalho (Moraes, 1999). O quarto é sub-estimador de todos os outros, sendo sempre menor ou igual a eles. Se for usado um sub-estimador muito baixo, a ramificação - arborescência - fica grande demais, i.e., esta heurística ruim começa a "brigar" com a característica ótima do **A***.

Perceba que o estimador não leva em conta o custo do terreno! É como se todo ele tivesse custo igual, uniforme. É exatamente isto que faz o Melhor Primeiro – que é a própria parcela $\hat{h}(n)$ do **A*** - não se preocupar com peso, rumando sempre em direção ao alvo. Isto é que "puxa" a parte Dijkstra – $g(n)$ - do **A*** em direção ao alvo.

Com tudo isto exposto, percebe-se que uma qualidade de um caminho e a eficiência da busca para o encontrar são funções do método de busca e da heurística estimadora empregados (Moraes, 1999).

2.9 Conclusões Sobre Otimização por Buscas em Grafos

O **A*** é um algoritmo de busca que colabora na resolução de muitos tipos de problemas (Moraes, 1999). Ele resolve:

* problemas de otimização de uma variável num espaço de estados;
* problemas de modo admissível e ótimo, dado um grafo já montado. Para se montar o grafo, é necessária uma discretização, e nela o **A*** não tem a menor interferência: se ela não for bem-feita, a resposta não será ótima, talvez nem será boa.

Como é possível perceber, há muitos problemas que o **A*** não resolve (Moraes, 1999). Exemplos seguem:

* problemas de difícil discretização, pois o grafo já chega a ele "maculado" pela discretização;
* problemas de otimização de n variáveis simultaneamente (otimizar n-upla, vetor, como faz o *Simulated annealing*);
* busca por vários estados (não somente um) desejáveis, sendo que tais estados podem ser encontrados em qualquer ordem. O

caixeiro-viajante é um exemplo. Pode-se, em teoria, fazer o **A*** buscar todas as possibilidades de ordem. Por simples análise combinatória percebe-se que mesmo para o caso de alguns poucos pontos, o tempo de busca seria muito grande, inviabilizando o **A***.

Parece estar claro que o **A*** não é o ideal para todos os casos com que se pode deparar no planejamento de caminhos de baixo custo: ele é útil quando a decomposição espacial se faz facilmente, i.e., quando há um conjunto finito de lugares em que o robô pode se encontrar, como se o ambiente fosse um "tabuleiro de xadrez".

Quando o espaço é contínuo e de bordas recortadas (sejam tais bordas as fronteiras do espaço ou dos obstáculos), entretanto, o **A*** não é o método ideal de solução, pois sua resposta perderá qualidade devido à discretização. Assim sendo, faz-se necessário o estudo de um método mais robusto. Esta foi a motivação para o estudo do *Simulated annealing* e sua aplicação ao planejamento de trajetórias.

3. O *SIMULATED ANNEALING*

O *Simulated annealing* – Recozimento Simulado – é um método de otimização probabilística que surgiu na década de 80 (Kirkpatrick *et al.*, 1983) para solucionar problemas combinatórios complexos (otimizando funções de centenas, até mesmo milhares de variáveis). Assim sendo, ele vem sendo utilizado para solução dos ditos "problemas de agendamento" (*scheduling problems, timetabling problems*), de "roteamento" de circuitos com integração em larga escala e escala muito larga (LSI e VLSI, respectivamente), de processamento de imagens e de um problema complexo muito tradicional, o Problema do Caixeiro-viajante.

Segundo Lutfiyya et al. (1992), "O *Simulated annealing* é especialmente atrativo quando as funções a serem otimizadas não são suaves, i.e., têm muitos mínimos locais...", pois ele não encontra problemas em fugir deles.

3.1 Origem

A idéia que originou esta meta-heurística foi o Algoritmo de Metropolis, da Físico-química, mais precisamente o Algoritmo de Montecarlo. Metropolis era, originalmente, um modelo computacional para simular o processo de recristalização de átomos num metal durante o recozimento. Recozimento é um processo térmico de lento resfriamento que visa a diminuir as tensões residuais internas de metais.

3.2 Motivação

Segundo Lutfiyya et al. (1992), para muitos problemas práticos ou teóricos, o objetivo é escolher a "melhor" solução dentre um grande número de soluções candidatas ou soluções espaciais. Tais problemas são tipicamente conhecidos como problemas de otimização combinatória, sendo formalizados como um par *(Estados,C)*. *Estados* é o conjunto de possíveis configurações (também chamado de espaço de busca) e, C, uma função - custo, $C : S \Rightarrow \Re$. Tal função associa um número real a cada configuração. Assume-se que, quanto mais baixo o valor de C, melhor é a configuração correspondente (do ponto de vista de otimização). O problema, assim, achar uma configuração para a qual C tem seu mínimo valor, i.e., uma configuração ótima j_0 que satisfaça

$$C_{ot} = C(j_O) = \min_{j \in S} C(j) \qquad (4)$$

em que C_{ot} denota o custo ótimo, ou seja, mínimo.

Origem Física

Segundo Lutfiyya et al. (1992), em qualquer substância, as moléculas possuem diferentes níveis de energia, distribuídos em valores discretos, sendo o estado menos energético, i.e., de menor energia, chamado de estado fundamental. Isto permite que uma dada substância tenha muitos macro-estados, em diversos níveis de energia. O processo natural de troca de energia entre moléculas faz com que o sistema tenda à configuração com o mínimo possível de energia. A observação deste comportamento sugeriu a aplicação de uma simulação deste processo para a otimização de funções com muitas variáveis. Tudo isto será agora explicado com maiores detalhes.

O cerne do *Simulated annealing* é uma analogia com a Mecânica Estatística. Em tal ciência, que trata com números muito grandes de partículas, apenas o comportamento mais provável do sistema (a uma dada temperatura) é observado experimentalmente. Como o comportamento pode variar, em uma dada temperatura, a determinação do comportamento mais provável é feita considerando-se o comportamento médio (estatisticamente falando) de uma coleção de sistemas idênticos. Nesta coleção, cada configuração, definida pelo conjunto de posições atômicas $r_1, r_2, r_3, ..., r_j$, é ponderada por seu fator de probabilidade de Boltzmann

$$e^{-E(rj)/kT} \qquad (5)$$

em que $E(r_j)$ é a energia da configuração j, k é a constante de Boltzmann e T é a temperatura. Estados de mínima energia (e configurações próximas a eles em energia) são de ocorrência extremamente rara dentre todas as configurações possíveis, mas eles dominam a baixas temperaturas, pois conforme T diminui, a distribuição de Boltzmann colapsa no nível (ou níveis) de menor energia.

O sistema, para minimizar sua energia, tende a alinhar todos seus *spins* (momentos angulares magnéticos) na mesma direção. Assim sendo, a configuração – estado – de mínima energia é muito simples, com todos os *spins* apontando em uma mesma direção. Tal estado é chamado estado cristalizado, pois de fato o sistema se cristaliza. No estado cristalizado, E atinge seu valor mínimo, e a distribuição de Boltzmann implica que isto ocorre numa temperatura baixa.

Na prática, entretanto, ter-se uma temperatura baixa não é condição suficiente, apesar de ser necessária. Experimentos que determinam o estado a baixa temperatura com energia mínima são feitos por um método chamado recozimento (*annealing*). Primeiro o sistema é levado a alta temperatura, que então é baixada lentamente. Deixa-se o sistema por longo tempo nas temperaturas próximas do ponto de congelamento (*freezing point*). Isto porque precisa haver tempo suficiente em cada temperatura para que os *spins* alcancem o equilíbrio de alinhamento já citado (polarizando-se próximos de +1 ou -1). Isto é chamado de equilíbrio térmico ou *quasi equilibrium.* Se a temperatura fosse baixada muito rapidamente, o sistema não teria tempo suficiente para chegar ao equilíbrio e, assim, o estado resultante poderia ter alta energia. Relembrando, está-se buscando estados de baixa energia.

Como já exposto, a distribuição de partículas num sistema, em uma dada temperatura T, segue a distribuição de Boltzmann. Tal probabilidade expressa a idéia de que um sistema em equilíbrio térmico em uma temperatura T tem sua energia distribuída probabilisticamente entre todos os diferentes estados de energia. Mesmo a baixas temperaturas, ainda há uma chance, mesmo que pequena, de um sistema estar em um estado de alta energia. Portanto, há uma chance correspondente de tal sistema sair de um mínimo local não-global de energia e prosseguir em sua busca por estados com menores energias.

A Vantagem do Simulated Annealing

O *Simulated Annealing* é capaz de evitar máximos locais e mínimos locais não-globais indo, algumas vezes durante o processo, para piores estados intermediários (isto é, qualquer estado que tenha custo maior que o anterior). A probabilidade disto ocorrer é p, valor que decresce com a diminuição da temperatura (i.e., enquanto o algoritmo progride, tendendo à solução ótima). O valor de p é dado por expressão que será exposta neste texto.

Enfoque

O enfoque deste trabalho utiliza trajetórias poligonais cujos parâmetros são as coordenadas de seus vértices.

Seja τ uma trajetória poligonal bidimensional de q segmentos cujos vértices são $\{(x_0, y_0), (x_1, y_1), \cdots, (x_q, y_q)\}$. Os pontos (x_0, y_0) e (x_q, y_q) são, respectivamente, posição-início e posição-objetivo do problema de planejamento de trajetória e, assim,

comuns a todas as trajetórias, então as trajetórias podem ser representadas por vetores menores, que contêm apenas o que se chama neste trabalho de pontos intermediários, i.e., $\mathbf{x} = \{(x_1, y_1), \cdots, (x_{q-1}, y_{q-1})\}$. Note que $q+1$ é o número total de pontos, sendo apenas $q-1$ pontos intermediários; os outros pontos, S e G, podem ser representados por (x_S, y_S) e (x_G, y_G) em lugar de (x_0, y_0) e (x_k, y_k).

O Algoritmo: Funcionamento Básico

O laço principal do algoritmo gera uma versão modificada da trajetória adicionando vetores aleatórios aos pontos intermediários. Se a nova trajetória tiver um custo menor que a anterior, é imediatamente aceita; caso contrário, há ainda uma probabilidade de ser aceita, e é isto que torna o **SA** robusto.

Este é o cerne do algoritmo em linguagem estruturada didática baseada em PASCAL:

```
begin (1)
  S ← solução inicial
  T ← temperatura inicial T0
while (critério de parada não satisfeito)
  begin (2)
    while (não chegou ao equilíbrio)
    begin (3)
      S' ← vizinho de S ao acaso
      δ ← Custo(S') - Custo(S)
      Probabilidade := min(1, exp(-δ/T))
      se (random(0,1) <= Probabilidade) então
            S ← S'
    end (3)
atualizar T de acordo com a função de recozimento
  end (2)
informar melhor resposta como saída
end (1)
```

Função de Avaliação e Cálculo do Custo

Já se citou no texto que o enfoque é tentar minimizar a função de avaliação, que é a função que estima o custo total de se atravessar o terreno, desde o ponto S até o ponto G. Assim sendo, é interessante introduzir o conceito geral de função de avaliação.

A função de avaliação é, no caso geral, uma função usada para estimar ou calcular o quão "boa" é uma solução. Isto permite que o algoritmo utilizado, qualquer que seja, compare-a com outras e então tome suas decisões. A definição de "boa" é feita exatamente pela função de avaliação: o projetista, baseado nas características do problema em questão, define tal função como um estimador heurístico ou mesmo se utiliza de cálculos exatos, se possível.

Neste trabalho a função de avaliação é de fácil compreensão: uma soma infinitesimal dos custos dos terrenos atravessados. Como se escolheu o *Simulated Annealing*, que não necessita de discretização de espaço, o custo poderia ser, no caso geral, uma função Real de duas variáveis, do tipo:

$$f : \Re^2 \Rightarrow \Re \qquad (6)$$

Seriam essas duas variáveis aquelas que definem a posição (x e y, no caso bidimensional). Nos exemplos didáticos utilizados para se obter imagens da tela para este trabalho, um ambiente binário foi usado: custo unitário para se atravessar um terreno livre e custo vinte para se atravessar um terreno com obstáculo. A função de custo é, em tais casos:

$$Custo = Custo(x, y) = 20 \quad (x^2 + y^2 \leq R^2) \qquad (7)$$

Em que R é o raio do obstáculo presente no ambiente estudado.

Neste caso, o custo total da trajetória seria a integral de linha – do ponto-início S ao ponto-objetivo G – do custo do espaço atravessado. Esta integral, por sua vez, seria a soma das integrais dos segmentos da trajetória poligonal: de S ao ponto *1*, de *1* a *2* ... de *q-2* a *q-1* e de *q-1* a G.

Procedimento

Inicialmente, uma trajetória aleatória é gerada (ou um conjunto de trajetórias aleatórias, na implementação usual, da literatura). A cada iteração gera-se uma nova trajetória, denominada trajetória candidata, adicionando-se a um ponto da trajetória um pequeno valor aleatório no eixo x e um valor, independente do primeiro, no eixo y. Tal valor é uniformemente distribuído dentro de *[-lado,+lado]*, intervalo em que *lado* é o lado do quadrado que é a área de novos possíveis pontos candidatos, como se vê nas Figuras 5 e 6. Se a adição gera um ponto que resulta numa trajetória candidata que é mais barata em termos de custo que a anterior, esta solução candidata substitui a anterior. Caso contrário, ainda há uma chance da substituição ocorrer: um

número aleatório é gerado entre 0 e 1; se for menor do que uma probabilidade de aceitação calculada pela Lei de Boltzmann, então a trajetória anterior é substituída pela nova. Caso contrário, a trajetória candidata é descartada.

A Lei de Boltzmann aplicada ao caso deste trabalho tem como expressão

$$p = e^{-(CustoCandi\ dato - CustoAtual\ )/T} \tag{8}$$

em que *Custo Atual* é o custo da última trajetória aceita (ou da trajetória inicial, se for a primeira iteração), *Custo Candidato* é o custo da trajetória-candidata e T é o *parâmetro de temperatura*, calculado pela função de recozimento. A constante de Boltzmann, *k*, não é utilizada porque já se encontra inclusa nos outros termos. Quando é utilizada em **SA**, é somente como um parâmetro de ajuste do algoritmo.

Sendo feita ou não a substituição, haverá uma nova tentativa, caso o algoritmo não tenha terminado. Tal tentativa será feita escolhendo-se aleatoriamente qualquer ponto intermediário da trajetória.

Funcionamento

Seja o problema de minimizar a função $F(x)$, na qual x é um vetor. O algoritmo começa com uma solução possível e aleatória x_0 para o problema, como mostra a Figura 4. Em seguida, a cada iteração soma-se à solução um vetor x_a. Tal vetor possui apenas um componente não-nulo, que é aleatório e restrito a determinada amplitude máxima, como pode ser observado nas Figuras 5 e 6.

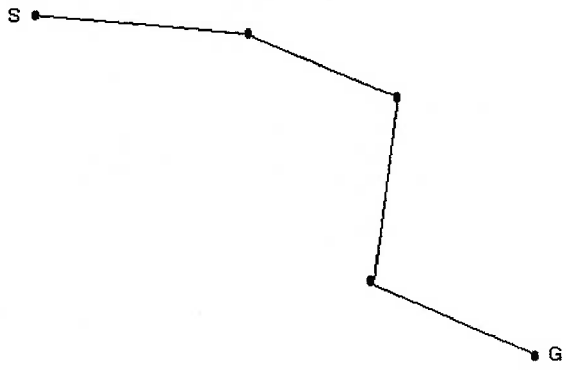

Fig. 4 . Uma trajetória poligonal, usada como solução inicial x_0.

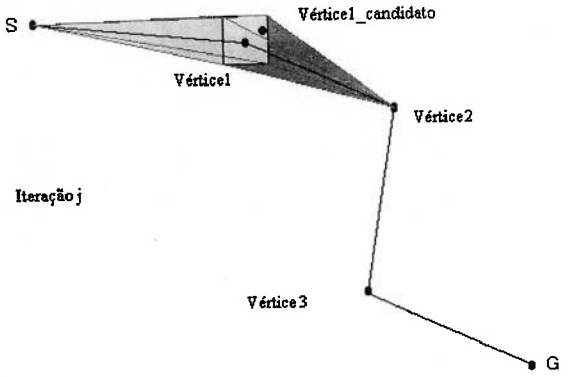

Fig. 5 . Possíveis variações aleatórias da trajetória na iteração j.

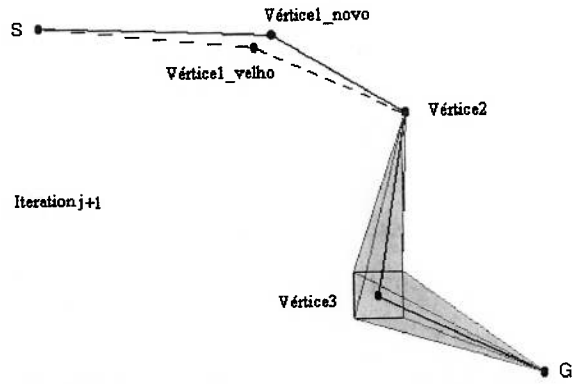

Fig. 6 . Iteração j+1: mudança aceita e um novo ponto usado para gerar a próxima solução candidata.

A Figura 5 mostra uma tentativa que gera uma solução candidata na *j*-ésima iteração: tenta-se um vértice aleatório próximo ao anterior (este é $vértice_1$, enquanto o vértice que se testa será o vértice candidato *1*, chamado $vértice_{1\_candidato}$). Sendo aceita ou não a solução candidata assim gerada, processar-se-á a próxima iteração, *j+1*-ésima. Será ela realizada à mesma temperatura que a anterior ou não, usando qualquer dos pontos intermediários, incluindo o próprio $vértice_1$, pois tal decisão será aleatória. Pontos intermediários são os pontos localizados entre os pontos fixos (que são o ponto-início e o ponto-objetivo ou ponto-alvo: *S* e *G*, respectivamente). Assim, o conjunto de pontos intermediários vai do primeiro ponto móvel, $vértice_1$, ao último ponto móvel, $vértice_{k-1}$. No exemplo acima, tratam-se dos pontos $vértice_1$, $vértice_2$ e $vértice_3$, pois há quatro segmentos na trajetória, resultando $q=4$.

A parte em questão da trajetória são dois segmentos que se juntam no ponto *vértice*$_1$: o segmento que vai do ponto inicial S até o ponto *vértice*$_1$ e aquele que liga este último a *vértice*$_2$. Como se pode ver na Figura 5, o próximo ponto candidato (i.e., o novo vértice *1*, representado por *vértice*$_{1new}$) será localizado na área cinza. Assim sendo, tal parte da trajetória pode ser encontrada em qualquer local dentro das áreas em ciano ou cinza. O *vértice*$_{1new}$ foi desenhado em vermelho, a Nordeste do ponto inicial. É exatamente a nova localização de *vértice*$_{1new}$ que define como ficarão, após tal iteração, os dois segmentos que o contêm. Eles passam a ser S para *vértice*$_{1new}$ e *vértice*$_{1new}$ para *vértice*$_2$.

É comum utilizar-se, para os oito sentidos associados às quatro direções básicas, a seguinte notação: pontos cardeais e colaterais. Define-se o Norte como o topo da página, o Nordeste como o canto superior direito, o Leste como a direita, o Sudeste como o canto inferior direito e assim por diante, até o Noroeste.

Se a tentativa for aceita, então a nova tentativa seria como mostrado na Figura 6: *vértice*$_{1new}$ está perto de sua posição anterior e a trajetória recém-aceita é desenhada em linha contínua, enquanto que a trajetória anterior (apenas a parte que foi alterada) é representada por uma linha tracejada. Uma nova tentativa será feita, desta vez usando o ponto *vértice*$_3$, aleatoriamente selecionado.

Se a soma $x_{candidato} = x_i + x_a$ resulta numa trajetória-candidata cujo custo é menor ou igual ao da trajetória atual, substitui-se a atual, então $x_{i+1} = x_{candidato}$. Caso contrário, ainda assim há probabilidade p de que a atual seja substituída por tal candidata; p é função do quão pior – que a solução atual – é a solução candidata. Definindo:

$$\Delta = F(x_{candidato}) - F(x_i) \qquad \textbf{(9)}$$

Então a probabilidade é definida como:

$$p = e^{-\Delta / T} \qquad \textbf{(10)}$$

na qual T é um parâmetro do processo (fisicamente, no recozimento, é a temperatura do sistema). Esta função é motivada pela distribuição de Boltzmann a respeito do estado energético das partículas de um sistema.

O parâmetro T decresce com o progresso do algoritmo, então há uma simulação do "resfriamento" do sistema desde a temperatura inicial até que uma condição de parada seja alcançada.

Variações do algoritmo incluem o modo como o vetor aleatório x_a é gerado (tipicamente, utilizando distribuição uniforme) e o modo como T decresce e o critério de parada. Tudo isso será mais profundamente discutido no decorrer deste trabalho.

4. DETALHANDO O SIMULATED ANNEALING

Primeiramente, as funções existentes de recozimento são definidas e descritas. Notação: utiliza-se neste trabalho o termo função de recozimento ao invés de função de resfriamento, por ser o primeiro termo mais geral que o segundo: a literatura em sua grande maioria pesquisou otimizações para o resfriamento do sistema, i.e., apenas trabalhavam com funções monotonicamente decrescentes da temperatura. Neste trabalho, a temperatura pode também ser aumentada. Há trabalhos que utilizam até mesmo temperatura constante durante todo o processo.

Em segundo lugar, são expostas heurísticas de melhoria de desempenho e qualidade implementadas neste trabalho. O mesmo será feito para heurísticas propostas e implementadas neste trabalho.

Tais heurísticas são usadas para melhorar não somente o desempenho (leia-se a eficiência, a convergência, o tempo para calcular a trajetória, o tempo para dar a solução ao problema proposto) como também a eficácia do algoritmo (leia-se qualidade das soluções por ele encontradas).

4.1 Funções de Recozimento

Função de recozimento é a função que definirá a evolução da temperatura com o passar das iterações do algoritmo, i.e., com o progresso deste.

A função de recozimento mais utilizada atualmente é o resfriamento geométrico, *geometric cooling* (e suas evoluções), que possui a seguinte fórmula geral (Preiss, 1997; Cohn e Fielding, 1999):

$$T_{i+1} = T_i \cdot \alpha \qquad (11)$$

Na expressão, i é o índice da temperatura e Ti é a i-ésima temperatura (a temperatura inicial utilizada é T_0, i.e., $T(i=0)$). O fator α é o multiplicador que ditará o ritmo do resfriamento, sendo denominado fator de resfriamento (*cooling factor*); tem ele valor positivo e menor que o unitário.

O resfriamento geométrico e suas evoluções serão descritos neste trabalho, tendo sido eles implementados no programa **SAtest**.

Outra função de recozimento bastante utilizada é o resfriamento logarítmico, *logarithmic cooling*, de expressão (Cohn e Fielding, 1999):

$$T_i = c/(\log(i + i_o)) \qquad \textbf{(12)}$$

Na expressão, *Ti* é a *i*-ésima temperatura, *c* é um valor ajustável e i_o é um inteiro positivo. Perceba que a expressão da temperatura atual não se relaciona à temperatura anterior como no caso anterior, mas sim a dados fixos e ao índice da temperatura, *i*.

Ambas as funções, geométrica e logarítmica, foram implementadas, testadas e comparadas neste trabalho. Implementou-se também a função linear de recozimento, explicada logo à frente.

Qual função de recozimento é a melhor? Esta pergunta não tem resposta direta: mesmo quando se considera um único problema, precisa-se de análise mais profunda. No caso do TSP (problema do caixeiro-viajante), Fielding (Cohn e Fielding, 1999) descobriu que dependendo do tipo e do tamanho do problema seria esta ou uma outra função a mais adequada.

Este trabalho se preocupa com o caso da busca por trajetórias de baixo custo para robôs móveis. Para tal aplicação, a convergência da função logarítmica de recozimento é mais lenta do que aquela da geométrica. Hajek (1988) provou que a primeira acha o mínimo global, mas apenas após um número muito grande de iterações. Na verdade, a prova foi feita para tal número tendendo a infinito. Ambas têm comportamento similar: tendem assintoticamente ao custo mínimo global quando a temperatura tende assintoticamente a zero. A função logarítmica de recozimento, entretanto, leva mais tempo que a geométrica para alcançar um mesmo pequeno valor dado.

Embora a convergência tenha sido provada apenas para o recozimento logarítmico, os pesquisadores têm preferido o resfriamento geométrico acreditando no empirismo, já que o geométrico gera uma solução próxima da ótima – ou mesmo ótima – num intervalo de tempo bem menor (Cohn e Fielding, 1999). Soluções não-convergentes, quase-ótimas foram adotadas em trabalhos relacionados a **SA** (Lundy e Mees, 1986; Aarts e Laarhoven, 1985), e mesmo no original, de Kirkpatrick (Kirkpatrick *et al.*, 1983), como será discutido abaixo. É importante frisar que a convergência da função logarítmica só é provada para o caso em que o número de iterações tende a infinito e *c* é maior ou igual à constante canônica *d**, que é a maior

profundidade de mínimo local que não seja mínimo global (Cohn e Fielding, 1999). A prova e a identificação do valor canônico foram feitas por Hajek (1988).

Outra função de recozimento que é uma função de resfriamento monotônico é o resfriamento linear, originalmente proposto por Kirkpatrick no artigo que introduziu o *Simulated annealing* (Kirkpatrick *et al.*, 1983); sua fórmula geral é:

$$T_{i+1} = T_i - T_{decremento} \qquad \textbf{(13)}$$

Na expressão acima, $T_{decremento}$ é o decremento da temperatura, i.e., o quanto se baixará a temperatura em cada oportunidade de resfriamento. Tal valor é fixo e pré-determinado.

O resfriamento linear também foi implementado neste trabalho e usado para as comparações de desempenho.

Exemplos de funções de recozimento em que não se faz resfriamento incluem fervura (trabalhar a altas temperaturas) e temperatura constante (Cohn e Fielding, 1999). Há ainda aqueles métodos que se utilizam de reaquecimento. Pelo exposto, a expressão geral *annealing schedule* (função de recozimento) mostra-se mais adequada para o caso geral, quando comparada com o termo utilizado pela literatura, *cooling schedule* (função de resfriamento), surgido quando ainda só havia funções monotonicamente decrescentes de evolução da temperatura com o progresso do algoritmo.

4.2 Evoluções do Resfriamento Geométrico

As melhorias para a função de resfriamento geométrico são introduzidas abaixo, uma a uma. Explica-se também a união de suas características desejáveis.

O Simulated Annealing *Adaptativo*

A primeira das evoluções da função geométrica é o ASA, *Adaptive Simulated Annealing*, Recozimento Simulado Adaptativo (Elmohamed, Fox, Coddington, 1998; Huang *et al.*, 1986): é ele um resfriamento retroalimentado, baseado no controle em malha fechada do recozimento geométrico. Tal controle faz o resfriamento se tornar mais rápido ou lento: uma taxa variável de mudança de temperatura é usada, isto é, α passa a ser variável, dependendo de critério de controle exposto a seguir.

O desempenho do algoritmo aumenta com tal melhoria, pois pode o processo progredir com decrementos maiores da temperatura quando isto for compatível com a manutenção da qualidade da solução, de seus estados intermediários, e a heurística de teste do valor do α cuida de tal compromisso. A qualidade da solução pode até mesmo aumentar, pois o resfriamento mais lento em momentos de necessidade possibilita a fuga de mínimos locais não-globais por parte do processo. O ASA foi implementado neste trabalho, e os testes confirmam a teoria. Em tal implementação, a taxa de resfriamento α varia entre 0,90 e 0,99 ao invés de ser constante (geralmente 0,95 para taxas fixas de resfriamento geométrico).

Qual é a informação usada na retroalimentação para decidir quando alterar ou não α, e como? É a variância de custo de dez **SA** que são processados em paralelo (simultaneamente e dependentes ente si). Fala-se aqui de variância em sentido estatístico. Ela é computada e usada como uma estimativa do calor específico C_H do sistema no conjunto de estados existentes: quanto maior a variância, menos "bem comportado" está o sistema, e menor é a tendência de que tenha boa convergência se mantido o ritmo do recozimento. A obtenção da variância se faz com estados todos de mesma temperatura, pois todos os **SA** estão na iteração de mesmo número, progridem juntos, em paralelo. Assim, centenas, milhares de iterações ocorrem no **SA** de número um - idem para os outros nove - até que se mude a temperatura, o que ocorre para todos os **SA** ao mesmo tempo. Ao fim de todas as iterações daquela dada temperatura (i.e., na iminência de se mudar a mesma), todos os custos nela obtidos são utilizados no cálculo da variância. Tal cálculo serve para que se decida se haverá mudança na taxa de recozimento α ou não. Se sim, o novo valor de α será mudado em 0,01, um valor pequeno quando comparado ao de α (0,90 a 0,99). Isto para que as mudanças de ritmo no recozimento sejam sempre suaves. Tal procedimento se repete ao fim das iterações da temperatura seguinte e assim por diante até o fim do processo.

Qual a mudança que se impõe ao sistema baseada na variância recém-obtida? Em primeiro lugar é preciso que se entenda que qualquer que seja o valor da variância calculada, a temperatura é mudada. Assim, a decisão tomada baseada no valor da variância terá efeito somente na nova temperatura, ou seja, a temperatura seguinte no recozimento. Que efeitos são estes? Os seguintes: se os resultados dos

SA de uma dada iteração são muito diferentes entre si, o resfriamento fica mais lento (ou mesmo é substituído, por certo tempo, por reaquecimento, como se verá em seguida). O que é esta "diferença"? Julga-se que os estados encontrados estão com grande diferença entre si quando apresentam uma variância alta ou, em termos químicos, uma entropia alta.

Mais detalhadamente: se a variância do custo for menor que um valor-limite - que depende de ajuste fino, por parte do projetista, para cada problema -, α é considerado um valor aceitável para aquele ponto do processo. Assim, é mantido seu valor, e a temperatura será alterada exatamente como o foi na mudança anterior. Pode-se até diminuir α, caso em que o resfriamento se dará mais rapidamente. Se, pelo contrário, a variância do custo for maior que aquele valor-limite, conclui-se que o calor específico está muito alto e, assim, o sistema demorará para chegar ao equilíbrio. Assim, estima-se que o resfriamento esteja ocorrendo mais rapidamente que o desejável para aquele ponto do processo, e deve ser tornado mais lento: α é, assim, aumentado, chegando mais próximo do valor unitário, o que diminui o ritmo do resfriamento, mantendo o sistema por mais tempo naquela faixa de temperatura, como desejado.

Neste trabalho foram desenvolvidas algumas regras para alimentar o controle de malha fechada da taxa de recozimento; o algoritmo ainda é o ASA, mas este trabalho introduz o seguinte critério para controlar a taxa de resfriamento: o algoritmo usa apenas um exemplar, uma ocorrência de **SA** para calcular a variância entre os custos do sistema em uma dada temperatura. Os outros autores usam um número fixo de **SA** simultâneos, por exemplo dez, como visto acima. Para se manter uma quantidade boa de amostras de custos (para calcular a variância), faz-se neste trabalho mais iterações por temperatura. Assim, este trabalho resfria mais lentamente que os outros – o que é bom para convergência - e usa a mesma ordem de grandeza de valores de custo para calcular a variância, o que valida seu valor frente às implementações tradicionais do ASA.

Percebeu-se, assim, neste trabalho, que a idéia acima (usar apenas um **SA**, mas com mais iterações à mesma temperatura) é tão robusta quanto as propostas em outros trabalhos (Elmohammohamed,ngton, 1998; Huang *et al.*, 1986). Tais trabalhos usam alguns **SA** em paralelo, porém menos iterações por temperatura em

cada um deles (quando comparados ao que se implementou neste trabalho). Assim, utiliza-se quantidades semelhantes de valores de custo para se calcular a variância de custo (o estimador do calor específico do sistema).

Assim, a primeira regra desenvolvida neste trabalho para atualizar a taxa de recozimento é a comparação supracitada da variância com uma variância-limite. Esta primeira idéia é a utilizada por alguns autores que se utilizam de taxa variável de resfriamento (seja com ou sem reaquecimento). A nova idéia, introduzida neste trabalho, é exatamente trabalhar, adicionalmente, com uma segunda condição, que leva em conta o percentual de aceitação de tentativas, de soluções candidatas. Tal regra adicional é: se as soluções candidatas geradas numa dada temperatura têm um alto percentual de aceitação de estados candidatos (digamos maior que 80%), α também é diminuído (como o seria no caso da pequena variância). Perceba que para que haja diminuição de α (aceleração do processo de resfriamento), deve ocorrer a situação prevista na primeira e/ou na segunda regra.

Quando a variância está alta, percebe-se que o sistema tende a demorar para chegar ao equilíbrio (dentro de uma dada temperatura). Assim, a ação que se deve tomar é providenciar essa demora requerida pelo sistema, seja fazendo-se mais iterações na mesma temperatura, seja diminuindo a velocidade do resfriamento - isto é, fazendo o decremento de temperatura ficar menor do que seria para um resfriamento geométrico comum (Cohn e Fielding, 1999).

A lógica para as regras acima trabalhando juntas segue: quando a variância é considerada baixa (ou quando a taxa de aceitação está alta), pode-se perceber que o resfriamento poderia estar mais rápido, pois o sistema está bastante "comportado", chegando rapidamente ao equilíbrio (dentro de uma dada temperatura). Assim, ele convergiria mesmo com um resfriamento mais rápido, o qual requereria menos computação. Tal aceleração do resfriamento se providencia com redução de α. em um valor de 0,01.

O procedimento citado é reversível, i.e., o resfriamento pode ser tornado mais lento, voltando α ao valor anterior ou mesmo a um maior (levando, por exemplo, a um resfriamento mais lento que o inicial, ou mesmo a uma condição de reaquecimento). Assim Assim se.Ulizando a mesma lógica acima, de duas condições: α é aumentado em 0,01 quando se percebe que o resfriamento está muito rápido (seja

por estar a variância dos custos muito alta ou a taxa de aceitação muito baixa). Em qualquer destes dois casos, conclui-se que o sistema está ficando pouco "comportado", tornando-se desordenado, demorando a estabilizar (dentro de uma dada temperatura). Matematicamente falando, "pouco comportado" quer dizer que as várias soluções subseqüentes, obtidas àquela temperatura, estão muito diferentes entre si em termos de custo.

Vale perceber que, para o caso da variância alta, tem-se o sistema variando muito seus valores de custo naquela região (o seu gráfico ficaria muito "rugoso", cheio de vales e picos). No caso dos baixos índices de aceitação, estima-se que isto se dá porque boa parte das tentativas – soluções candidatas - resulta em energias significativamente mais altas que as anteriores, o que pode significar que a região contém um mínimo local (global ou não), e assim se justifica a desaceleração do resfriamento – e até o reaquecimento que, como já citado, auxilia o sistema em sua fuga de tais situações indesejáveis, os mínimos locais não-globais.

Deste modo, o ASA foi introduzido acima, sendo considerado a primeira melhoria para o resfriamento geométrico básico.

Reaquecimento

A segunda melhoria aplicada neste trabalho sobre a função geométrica é o reaquecimento. Ele se presta a aumentar a robustez do processo: com o reaquecimento, o processo tem aumentada sua capacidade de escapar de mínimos locais não-globais. A melhoria de desempenho sobre o Recozimento Geométrico já havia sido obtida pelo ASA

Diz-se que o reaquecimento é interessante por facilitar que a busca saia de poços de custo indesejáveis, i.e., mínimos locais não-globais. Mas como diferenciá-los de mínimos globais? O processo fica iterando num mínimo local não-global qualquer e, com o reaquecimento, ele "escapa" de tal situação indesejável. Isto porque com o reaquecimento progressivo – α é aumentado de centésimo em centésimo - o sistema passa a aceitar cada vez mais as situações em que há aumento de custo com relação ao estado anterior (lembrando: a probabilidade de tal aceitação aumenta com o aumento da temperatura). Assim sendo, o processo "escala de volta" o poço de custo. No caso de tal extremante ser um mínimo global, o critério de

parada normal do algoritmo terminaria o processo no próprio (tendo antes tentado escapar do suposto mínimo local não-global).

A limitação do reaquecimento se dá pelo número máximo de vezes que o ciclo reaquecimento/resfriamento pode ocorrer em seguida – pode-se utilizar um limite de três vezes por exemplo. É interessante limitar também o valor de α para que, a exemplo do resfriamento, o reaquecimento ocorra de modo suave: um reaquecimento excessivo leva a queda de desempenho. O valor usado no programa implementado foi 2.

Estudou-se dois tipos de reaquecimento: o primeiro é o Reaquecimento em Função do Custo, *Reheating as Function of Cost*, RFC (Elmohamed, Fox, Coddington, 1998), baseado na idéia original de reaquecimento (Abramson, Krishnamoorthy, Dang, 1997). O segundo – que está presente no programa gerado neste trabalho – é o Reaquecimento Geométrico, RG (Abramson, Krishnamoorthy, Dang, 1997).

O que os dois têm em comum: no ASA, com base na variação dos custos dos estados à mesma temperatura a velocidade de resfriamento era diminuída ou aumentada. No reaquecimento, a idéia é semelhante, mas o sistema pode até mesmo ser reaquecido.

A diferença entre o RFC e o RG é o cálculo de quanto se deve reaquecer é mais elaborado – e computacionalmente custoso - no RFC que no RG, pois no primeiro são feitos cálculos adicionais após os da variância para calcular o reaquecimento em si: usando-se dez **SA** em paralelo (como usa a literatura) ou apenas um (como proposto aqui), a variância é calculada e usada para computar C_H (calor específico), o qual é usado para calcular o aumento na temperatura, o reaquecimento (Elmohamed, Fox, Coddington, 1998; Abramson, Krishnamoorthy, Dang, 1997).

Assim sendo, preferiu-se implementar, aqui, o reaquecimento mais simples, de Abramson, Krishnamoorthy e Dang (1997), feito de modo similar ao ASA (variação do valor de α), mas "simétrico" a ele com relação à unidade, i.e., permite-se o valor de α superar o valor unitário. Assim, seu valor pode variar não somente de 0,50 a 0,99 (como era no ASA), mas de 0,50 a 2.

4.3 Comparações de Desempenho e Qualidade das Funções de Recozimento

Uma solução que seja ao menos próxima da ótima é necessária no caso do planejamento de locomoção de robôs, pois o tempo de execução, da movimentação em si, é geralmente significativamente maior que aquele de planejamento, e o primeiro poderia ser muito aumentado com uma solução de baixa qualidade, assim aumentando também o tempo total (que é a soma dos tempos de planejamento e execução).

Façam-se, assim, duas comparações: a primeira, ASA com RG comparado à função que torna o método mais eficiente (função linear de recozimento). Depois, ASA com RG comparado à função mais eficaz (função logarítmica de recozimento).

A primeira comparação se faz com um **SA** bastante rápido e simples, tanto na função de recozimento quanto no critério de parada: usa tal **SA** um resfriamento linear do sistema, com critério de parada tal que se termine o processo a uma temperatura T_F pré-fixada. Este algoritmo gerará, em muitos casos, uma solução de qualidade média ou mesmo ruim; o tempo adicional necessário para chegar a uma solução ótima – ou próxima disto – usando ASA com RG compensa no que se refere ao tempo total e ao custo de movimentação economizados por tal solução.

Proceda-se, agora, à segunda comparação: ASA com *RG* comparado ao método que garante solução ótima. Conclui-se, com a análise do compromisso supracitado, que a melhoria propiciada à solução pela função logarítmica não compensa seu tempo de processamento extremamente longo: lembre-se que a prova da convergência do recozimento logarítmico só foi feita para um número de iterações tendendo a infinito (Hajek, 1988).

4.4 Justificativa da Escolha da Função de Recozimento

Concluindo, tem-se que o recozimento linear tem maior desempenho, mas qualidade potencialmente insuficiente, enquanto que o ideal, logarítmico, cobra um preço grande demais por sua solução já provada ótima (Hajek, 1988): um número tendendo a infinito de iterações. Fica, assim, sendo ASA com RG a melhor solução para o caso tratado por este trabalho. Quando se fala em ASA, fica subentendido que se trata de função geométrica. A combinação em questão é também indicada para

muito outros casos, como se percebe por sua utilização pela literatura. Isto porque é a solução de melhor compromisso eficiência x eficácia, i.e., ela gera soluções de qualidade próxima ou igual àquelas da função logarítmica (tal qualidade é consideravelmente superior àquela da função linear). Tal alta qualidade é obtida em tempo muito menor que aquele requerido pela função logarítmica.

Em resumo, este trabalho implementou as seguintes funções de recozimento: geométrico melhorado, logarítmico e linear. O termo "geométrico melhorado" significa resfriamento geométrico básico com as duas melhorias: α adaptativo (ASA) e a possibilidade de reaquecimento, RG. Note que o RFC pode ser usado em qualquer função, não somente na geométrica; o RG, apesar de aparentemente fazer mais sentido na geométrica, pode ser usado em qualquer recozimento. Os benefícios do resfriamento geométrico são boa velocidade de convergência e uma solução que é ao menos quase-ótima.

Tenha-se em mente que o objetivo deste trabalho é planejar trajetórias de baixo custo entre dois pontos dados dentro de um espaço contínuo conhecido. O compromisso "desempenho do algoritmo *versus* qualidade da solução" leva à conclusão de que é ASA com RG a melhor solução, pois uma combinação do poder de malha fechada de ambos se faz presente: o desempenho do ASA aliado à robustez do RG. O desempenho também é ajudado pelo uso adicional das regras heurísticas e dos valores empíricos discutidos no Item 4.2.

4.5 Parâmetros Heurísticos

As regras heurísticas abaixo foram introduzidas para garantir que os algoritmos fossem robustos e convergiriam rapidamente para uma solução ótima ou quase ótima.

Temperatura inicial (T_0): temperatura inicial "T_0 deve ser escolhida de forma que a probabilidade de aceitação inicial [p_0] seja 0,8" (Preiss, 1997) ou 0,95 (Cohn e Fielding, 1999). Ambos os valores, além de outros intermediários (i.e., no intervalo por eles definido), foram testados. O primeiro foi usado.

Os valores recomendados para o fator de recozimento α e sua variação ao longo das iterações: muitos autores recomendam o uso de um intervalo fechado de 0,90 e 0,99 para o fator, ao longo de todo o processo, assim como em "Evidência

empírica sugere que um bom valor para α é 0,95..." (Preiss, 1997) e também no trabalho de Evers (1998). Foi, assim, tal valor utilizado neste trabalho como valor inicial do fator de recozimento.

Condições de equilíbrio em dada temperatura, i.e., condições necessárias para o processo ir para a temperatura seguinte: o critério mais simples é usar malha totalmente aberta: fazer um número fixo de iterações antes de se mudar a temperatura (Evers, 1998).

Há também o "Critério 2000": nele, o parâmetro T deve ser alterado quando 2.000 soluções candidatas forem tentadas – aceitas ou não, não importa – ou 200 soluções candidatas forem aceitas, o que quer que ocorra primeiro (Moraes, Martins e Tsuzuki, 2001). Este critério é uma adaptação do critério de parada de Schreiber, Schmitz e Richter (1998), que utilizava os valores 20.000 e 2.000, ao invés de 2.000 e 200, respectivamente. Tais valores eram maiores porque o recozimento utilizado naquele trabalho era o geométrico simples – com α constante -, e com valor de α igual a 0,5 (significativamente menor que aqueles usados neste trabalho). Sendo assim, o recozimento era bastante rápido, e um maior número de iterações por temperatura se fazia necessário.

4.6 Critério de Parada

Alguns critérios de parada do **SA** serão agora definidos, e o desenvolvido neste trabalho será introduzido.

A idéia original de Kirkpatrick, usada em seu resfriamento linear (Kirkpatrick *et al.*, 1983): o algoritmo termina quando um número fixo de iterações é feito a uma T_F pré-determinada. Mais tarde ele propôs um critério de parada que verificava existência de melhoria da solução com a evolução das iterações: o algoritmo deveria ser interrompido se, de acordo com dado critério de avaliação, não houvesse nenhum progresso notável após um pré-determinado número de iterações, pois o sistema alcançou o que Kirkpatrick chamou de *cold state* - ou estado frio. O valor considerado "notável" deve ser definido pelo projetista, para cada caso: deve ser um valor irrisório de progresso para aquela dada aplicação.

Critério de parada de resfriamento: quando dois resfriamentos (decrementos da temperatura, parâmetro T) consecutivos são feitos a partir do "Critério 2000" e

não há nova solução aceita, o sistema é considerado congelado (*frozen*): o algoritmo termina e a solução atual é aceita (Evers, 1998; Preiss, 1997).

Neste trabalho, o algoritmo termina se um número pré-definido de temperaturas consecutivas é processado com pequena ou nenhuma melhoria da solução. O termo "melhoria pequena" quer dizer: com uma melhoria de custo acumulada, ao longo daquelas últimas temperaturas, menor que um valor pequeno, também pré-definido (por exemplo, 1%).

4.7 Uma Consideração Importante Sobre a Escolha de Soluções Candidatas

Por que usar um ponto por vez, i.e., porque não adicionar um vetor inteiro de dimensão *q-1* de valores aleatórios para incrementar todos os pontos intermediários da trajetória?

Uma das condições do recozimento (físico e, portanto, também o simulado) é o lento prosseguimento do processo. No **SA**, ocorrer o processo lentamente não significa somente manter pequenos os decrementos ou incrementos de temperatura, mas também alterar a solução de modo quase infinitesimal quando se tenta uma solução nova, melhor, i.e., quando se gera uma solução candidata. Isto é conseguido quando se muda apenas um vértice por vez; entretanto, se todos os vértices são mudados ao mesmo tempo, mesmo que com pequenas mudanças em cada um, tem-se uma mudança grande na trajetória. Isto gera, em uma porcentagem grande demais das tentativas, soluções inaceitáveis, portanto rejeitadas, não aceitas. Tal rejeição diminui a velocidade de convergência do processo, pois a evolução do algoritmo se dá com aceitação de novas soluções, sejam elas melhores ou piores que a anterior.
Além disso, alterando-se apenas um vértice por iteração, minimiza-se os efeitos negativos do agrupamento (*clustering*), um fenômeno indesejável, detalhado no item 7.2.

4.8 Conclusão da Análise do SA e de suas Melhorias

Uma importante conclusão desta seção é que a função de recozimento, o critério de parada e a função de avaliação são partes cruciais para o desempenho do algoritmo e para a qualidade da solução por ele gerada.

Neste trabalho, testes exaustivos foram feitos: muita heurística (fórmulas e parâmetros) foi teorizada e também desenvolvida, sendo testada e depois avaliada. Comparou-se diversas implementações do **SA** umas com as outras; fez-se também comparações entre o **SA** e o que se tinha anteriormente: configuração espacial seguida de busca em grafos. As conclusões de tudo isto serão expostas ao fim do texto.

5. IMPLEMENTAÇÃO

Implementou-se, na linguagem de programação C++, um planejador de trajetórias para o caso bidimensional. Pôde-se, assim, capturar imagens do programa, imagens estas que são de especial interesse para a ilustração do exposto e são neste texto reproduzidas.

Para simular o "resfriamento" e o "aquecimento" do sistema, o parâmetro T é alterado com as iterações do algoritmo, de acordo com a função de recozimento escolhida.

5.1 Comparação do SA com o Outro Método

O computador utilizado para desenvolvimento e comparações de desempenho foi: processador Pentium II de 300 MHz, memória RAM de 256MB, Sistema Operacional Windows 98SE.

Por que utilizar o **A*** para se comparar com o **SA**? O **A*** e o **SA** são algoritmos com enfoques diferentes, pois têm interesses diferentes: o primeiro é mais usado para cálculos em tempo real, principalmente quando já se tem uma configuração espacial (a decomposição, discretização do ambiente de trabalho) pronta: a partir de tal ponto, uma simples e rápida busca em grafo ocorre: o tempo ótimo (tempo mínimo) de cálculo de uma trajetória admissível (para discretização e heurística dadas) é da ordem de unidades de milissegundo a dezenas de milissegundo para os muitos exemplos testados. Para os mesmos exemplos, no mesmo computador, o **SA** demorou um tempo da ordem de dezenas de segundos para achar solução, i.e., ele precisa de tempos de três a quatro ordens de grandeza maiores que o **A***.

Surge, então, a pergunta: "Por que o **SA**, então?". O **A*** é, além de admissível, ótimo. Se o **SA** traz como grande vantagem achar caminhos admissíveis (e o **A*** também), por que usá-lo, se o **A*** dá soluções tão boas quanto, mas em um tempo de processamento milhares de vezes menor? A resposta é que a resposta do **A***, na prática, normalmente não será tão boa quanto a do **SA**: este gera uma resposta mais precisa, pois o **A*** apenas entende como pontos (pontos possíveis, potenciais para a trajetória) aqueles que estão no centro dos "azulejos" (os quadrados ou

hexágonos usados na discretização do espaço). O **SA**, por sua vez, entende, como ponto viável para a trajetória, qualquer ponto do espaço contínuo que respeite a discretização do processador e do compilador utilizados.

Pensará o leitor: "Por que não usar, então, também no **A***, discretizações maiores, até que se chegue no limite do computador, i.e., ao 'quase-contínuo'?". A má notícia para o **A*** é que seu desempenho cai com o aumento da discretização: muitos mais azulejos serão criados para descrever com maior exatidão o espaço, e neles também deverá se processar a busca. Se o tamanho do lado do azulejo cai dez vezes, o número de azulejos eleva-se em cem vezes, e, assim, o mesmo ocorre com o número de nós do grafo em que se dará a busca (lembrando: cada nó do grafo representa um azulejo do espaço discretizado). Assim, um indesejável compromisso se instala quando se usa o **A***: robustez x desempenho. Robustez é necessária, e isto o **SA** tem muito mais que o **A***, já que o **SA** não precisa de configuração espacial (por trabalhar diretamente com o espaço contínuo).

Outra vantagem do **SA** em relação ao **A*** é a que ele tem em relação a qualquer algoritmo que se utilize de configuração espacial: este último encontra dificuldades para trabalhar em ambientes cuja decomposição espacial seja difícil. Por exemplo: um ambiente de superfícies (fronteiras e bordas de obstáculos) muito recortadas, como a costa de um país vista num mapa. Deve-se escolher uma de duas soluções nada encorajadores: um, faz-se o algoritmo de confiança especial perder um tempo grande decompondo tal ambiente e, assim, fará ele uma decomposição razoável ou boa. Assim, o algoritmo como um todo gerará uma solução aceitável, boa, ou mesmo excelente, mas a custo grande de tempo. Neste caso, a grande vantagem do **A*** - o desempenho - estará arruinada. Isto se dá tanto pela maior demora da configuração espacial quanto pelo maior tempo de que precisará o **A*** para buscar neste grafo maior. Dois, faz-se uma discretização comum, rápida do ambiente, gerando um grafo pequeno quando comparado com o da solução um. O **A*** inicia-se logo e todo o processo termina rapidamente, pois tanto configuração quanto a busca foram rápidas. Claro que a solução não ser tão boa.

Ora, quando se tem um ambiente topologicamente complexo, portanto, o projetista pode escolher o **SA** e obter uma solução demorada e ótima (ou quase-ótima) e de grande precisão. Ou ele escolhe o **A*** e se vê obrigado escolher entre a

qualidade da solução e o desempenho do processo. E provavelmente terá, ainda assim, uma solução menos precisa que aquela do **SA**.

Após tal argumentação pode voltar à tona novamente a pergunta: "Por que o **A***?". Ora, ele gera soluções de qualidade razoável ou mesmo boa em tempos milhares de vezes menores que aqueles do **SA**, como já exposto. Assim, ele foi essencial durante o desenvolvimento deste trabalho: houve a realização exaustiva de testes com ele antes de se usar o **SA** para a resolução do problema de planejamento de trajetórias.

5.2 Como o A* pode colaborar com o *Simulated annealing*

Pelas características reafirmadas logo acima (soluções de qualidade razoável em tempos menores que aqueles do **SA**), percebe-se que o **A*** é um bom candidato para ser o pré-processador gerador da solução inicial. Atualmente, ele é determinado aleatoriamente. Na verdade, é assim que a grande maioria das implementações faz. A idéia do pré-processador havia sido concebida neste trabalho, e uma heurística simples (que se utilizava de algumas características do ambiente em questão) havia sido projetada. Foi quando se teve acesso ao trabalho de Elmohamed, Fox e Coddington (1998). Nele, dois pré-processadores são introduzidos: um, complexo, inicial, é o exposto acima: o gerador de solução inicial. Embora não tenho sido usado o **A*** naquele trabalho, espera-se que a solução gerada por ele seja melhor que uma aleatória. O pré-processador inicial é usado apenas uma vez em cada utilização do **SA**: na geração da solução inicial. O outro pré-processador, de pequena complexidade, tenta fazer com que a escolha da próxima solução candidata tenha alguma "inteligência". Tal escolha é implementada com lógica meramente aleatória na maioria dos trabalhos. A heurística usada por Elmohamed, Fox e Coddington (1998) nesta segunda heurística era mais simples por motivos óbvios: tal etapa se processa milhares de vezes em cada utilização do **SA**: qualquer coisa de complexidade maior que "muito simples" diminuiria em muito o desempenho do algoritmo como um todo.

A idéia, assim, é usar o **A*** para gerar uma solução inicial que seja ao menos razoável para que possa o **SA** iniciar já "bem encaminhado" seu trabalho. Perceba que o tempo que o **A*** tomará para dar sua solução é milhares de vezes menor que

aquele que tomaria o **SA** para dar a solução para o mesmo problema. Assim, o **SA** iniciará sua busca com uma solução já bastante evoluída. De modo que o **SA** precedido de **A*** necessitará de muito menos iterações para chegar a uma solução de mesma qualidade. Isto em comparação com o **SA** sem **A*** como pré-processador.

Há outra grande vantagem na utilização de um pré-processador baseado em **A*** para escolha da solução inicial: já se recebe uma informação necessária ao início da busca via **SA**: o número de segmentos que deverá ter a trajetória poligonal. Isto ocorre porque a solução gerada pelo programa **AStar** é exatamente uma trajetória poligonal cujos segmentos podem estar nos eixos horizontal, vertical e um dos dois diagonais.

5.3 Quando o A* é suficiente

Vale a pena fazer uma importante ressalva: dependendo da qualidade da solução que se deseja para um dado caso e do tempo de que se dispõe para os cálculos que gerarão a solução, pode ser interessante fazer exatamente como Moraes (1999): discretizar o espaço em questão, calcular uma trajetória tão boa quanto possível com aquela dada configuração espacial (i.e., uma trajetória admissível) em tempo mínimo (nenhum outro algoritmo de busca em grafos é mais eficiente que o **A***, i.e., ele é ótimo) e se contentar com o resultado, que pode ser igual ou pior que o do **SA**. Novamente: não há um melhor algoritmo: há o algoritmo mais indicado para esta ou aquela situação.

Uma observação: com "uma trajetória tão boa quanto possível com aquela dada configuração espacial" quer-se dizer que a configuração espacial será um limitador de qualidade da solução: o **A*** mostrará a solução de melhor qualidade que se pode obter com dada heurística (função de estimativa de distância até o objetivo) e dada configuração espacial. Um outro modo de expor isto é dizer que o **A*** não consegue consertar heurísticas e configurações espaciais ruins: assim, uma busca **A*** bem implementada - em termos de estrutura de dados e também com uma boa heurística estimadora de custo -, que vá procurar valores num grafo que foi montado com uma configuração espacial deficiente, possivelmente não gerará boa resposta. O mesmo ocorre para o caso oposto: um grafo advindo de uma boa configuração espacial será percorrido por um **A*** bem estruturado, mas com uma heurística ruim

(por exemplo, regras ruins para tomada de decisão num programa jogador de xadrez).

Um exemplo de caso em que o **A*** é suficiente é o Exemplo 4, visto abaixo neste texto. Comenta-se, naquele exemplo, porque o **SA** é desnecessário – e o **A*** é suficiente – para o caso.

6. RESULTADOS

Os resultados serão apresentados por Estudo de Casos, que será feito resolvendo-se os exemplos propostos com a utilização dos dois programas em questão: **AStar** e **SAtest**.

6.1 Introdução ao Método e Objetivos dos Testes

Os programas utilizados nos testes são o **AStar** (cujo nome vem de **A***), que pode ser encontrado em CD junto ao trabalho de Moraes (1999) e o **SAtest**, produzido neste trabalho.

O espaço estudado no **SAtest** é um quadrado bidimensional de 300*x*300 unidades com obstáculos circulares, dado que as colisões de trajetórias poligonais com obstáculos circulares são facilmente computáveis. No caso do **AStar**, usa-se ambientes tão semelhantes a este quanto a discretização por quadrados do mesmo permitia. O custo para se atravessar um espaço obstruído é definido como vinte vezes o custo de se atravessar um espaço normal, tanto nos testes do **AStar** quanto nos do **SAtest**.

Os exemplos a seguir também tentam pôr o programa neste trabalho desenvolvido em situações delicadas. Se ele suceder, fica comprovada a validade do método em termos de qualidade e desempenho. Há, potencialmente, casos em que ambos os algoritmos apresentam boas soluções e casos em que este ou aquele é mais indicado, pois o outro método encontra dificuldades.

É pertinente a esta altura que se faça algumas observações a respeito de notação do programa **AStar** e de seu funcionamento, de modo que se compreendam os resultados por ele gerados. É claro que o interesse não é comparar ambos os programas: tal comparação é o meio, não o fim: o interesse verdadeiro é comparar o *Simulated annealing* com o **A*** para a aplicação a planejamento de trajetórias de robôs. Lembre-se que, como já exposto, o **A*** é o melhor dos métodos chamados *graph-searching techniques*, pois é admissível e ótimo.

Fez-se ícones para representar cada um dos dez tipos de terreno transponível e um para representar o intransponível. Os primeiros têm custo de travessia de um a dez, com incrementos unitários. Há também ícones para o ponto-início e o ponto-

objetivo. Foram os ícones projetados exatamente para dar uma idéia mnemônica ao mapa e também um aspecto visualmente claro; são eles representados na tabela I:

Tabela I: Ícones de custos

Tipo de Terreno	Ícone	Cor	Custo, Peso	Significado
Ponto-início			Qualquer	Ponto inicial, obrigatório
Ponto-objetivo			Qualquer	Ponto final, obrigatório
Terreno intransponível			Infinito ou muito grande	Parede, buraco, água etc.
Asfalto			1	O terreno menos custoso
Terra			2	Terreno de baixo custo
Vegetação Esparsa e Rasteira			3	Terreno de baixo custo
Textura Fixa			4	Terreno de baixo custo
Grama Rasteira			5	Terreno de médio custo
Madeira			6	Terreno de médio custo
Superfície Rochosa Irregular			7	Terreno de alto custo
Areia			8	Terreno de alto custo
Grama Alta			9	Terreno de alto custo
Poça d'água			10	O terreno mais custoso

Uma observação: para ambientes tais que nenhuma de suas duas dimensões ultrapassem vinte e nove azulejos, o **AStar** usa, para cada um desses azulejos, um dos ícones supracitados. Para trinta ou mais azulejos, utiliza-se cores ao invés dos ícones, por facilidade de visualização: são os onze ícones de diferentes custos um degradê de laranja, indo de branco (custo mínimo, unitário) a preto (custo "infinito"). Isto vale para trinta ou mais azulejos em qualquer um dos dois eixos, assim um mapa com doze por quarenta azulejos enquadra-se nesta categoria. Utiliza-se, ainda no mesmo caso, verde e vermelho para os pontos S e G, respectivamente.

Outra informação de interesse é o modo como se computam os custos de movimentação pelo *grid*: dividir-se-á os movimentos de um azulejo para seu adjacente em diagonais e não-diagonais (estes podem ser verticais ou horizontais). Para se ir de um azulejo a outro nos eixos horizontal ou vertical "paga-se" o custo do quadrado em que se entra, e só dele. Por que só se conta tal valor, e não aquele do ladrilho do qual se sai com aquele dado movimento? Simples: porque tal custo foi computado no passo anterior (pois naquele passo, o anterior, o robô entrou no azulejo do qual está saindo neste passo). Um movimento diagonal tem uma fórmula mais elaborada: tal movimento envolve quatro azulejos, não somente dois. Todos eles são utilizados no cálculo: faz-se uma média aritmética simples e se multiplica o valor por 7/5, aproximação de $2^{1/2}$, distância que separa os centros dos ladrilhos inicial e final daquele movimento unitário. É fácil perceber como tal estimativa introduz erro; nunca é demais lembrar que nada disto se faz necessário quando se utiliza o **SA**, como pode ser claramente visto na solução do Exemplo 1 gerada pelo programa **SAtest**, Figura 8.

6.2 Os Testes Comparativos em Si

Nesta seção, apresentam-se os testes; a partir deles faz-se comparações entre as soluções obtidas pelos dois métodos e sobre os métodos em si.

O Exemplo 1

No Exemplo 1 há um único obstáculo circular de raio igual a 60 unidades de comprimento, localizado no centro do espaço de busca.

Este é um exemplo de duas trajetórias localmente mínimas: as que passam ao "norte" do obstáculo e as que passam ao "sul". Além do mais, estas duas trajetórias

são simétricas com relação à reta que liga os pontos S e G, então não há diferença do ponto de vista de custo em se adotar uma ou outra; isto pode parecer bom, mas na verdade "confunde" o algoritmo, e por isto foi escolhido: as soluções alternar-se-ão, principalmente na primeira metade, dois terços iniciais do algoritmo: a trajetória atual está passando ao sul do obstáculo e se gera uma solução candidata ao norte dele, sendo ela aceita, por exemplo. Isto, intuitivamente se percebe, diminui a velocidade de convergência, pois não é óbvio ao algoritmo que solução é "boa" e qual é ruim. De modo contrário, os ambientes que possuem maior clareza (por exemplo, se fosse o obstáculo deste exemplo deslocado de um raio para a direita) facilitam a percepção por parte do algoritmo, ele não fica alternando entre passar ao norte ou ao sul e a convergência é acelerada. O segundo exemplo que será apresentado é ainda mais claro a respeito disto, já que usará três pares – ao invés de apenas um – causados por simetria.

Primeiramente mostra-se, nas Figuras 7 e 8, a resolução do Exemplo 1 pelo método **SA**, programa **SAtest**, com utilização de uma única trajetória de nove segmentos.

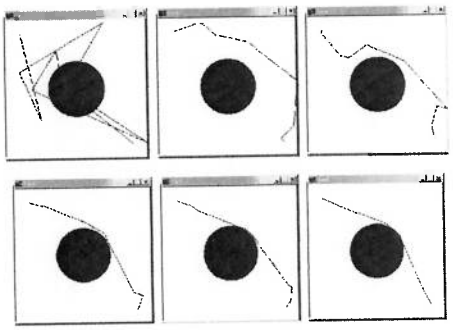

Fig. 7 . O progresso da busca por um caminho pouco custoso via *Simulated Annealing*, programa **SAtest**. O último quadro é a resposta.

É claramente perceptível, nos quadros da Figura 7, o progresso da busca: os cinco primeiros quadros mostram a evolução da qualidade da solução ao longo das iterações; o sexto e último quadro mostra a resposta final, i.e., o algoritmo já tendo convergido e o programa mostrando a tela gráfica da resposta.

A função de recozimento é, como já exposto, parte crucial do método; neste exemplo, a solução foi processada com o fator α diminuindo linearmente com as iterações, com um número máximo de 12000 iterações. O fator T alcançou o valor

zero na iteração de número 11000, assentando assim a trajetória em seu mínimo global.

Fez-se este ambiente simples com a função linear de recozimento para mostrar que ela pode ser suficiente para tais casos. Assim sendo, as funções mais complexas, como a geométrica e a logarítmica, seriam um excesso no caso: teriam pior desempenho com solução igual ou muito semelhante.

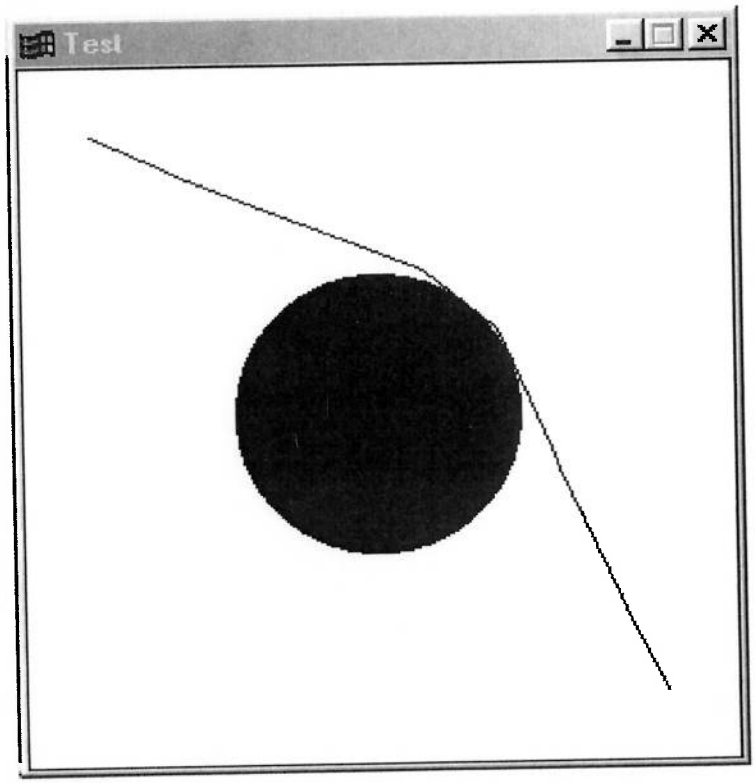

Fig. 8 . Resolução do Exemplo 1 pelo programa **SAtest**. A figura está na mesma dimensão daquelas geradas pelo **AStar**, para comparação visual.

O último quadro da Figura 8 é repetido agora em dimensões maiores para dar ao leitor um maior detalhamento e a ele facilitar a comparação visual com as três respostas geradas pelo **AStar** (Figuras 9, 11 e 14).

O processamento deste exemplo pelo **AStar** se deu com as seguintes discretizações: quadrados de dez (chamar-se-á discretização pequena ao longo do texto), trinta (discretização intermediária) e sessenta "ladrilhos" ou "azulejos" de lado (discretização grande), como se vê, respectivamente, nas Figuras 9, 11 e 14.

A fronteira de azulejos intransponíveis presente no ambiente (Figuras 9, 11 e 14) não é necessária ao funcionamento do programa, sendo meramente para ilustrar

ao usuário que ali se encontra o limite do espaço de busca. Pode-se fazer, no editor do **AStar**, ambientes sem tal fronteira e verificar o funcionamento normal do programa.

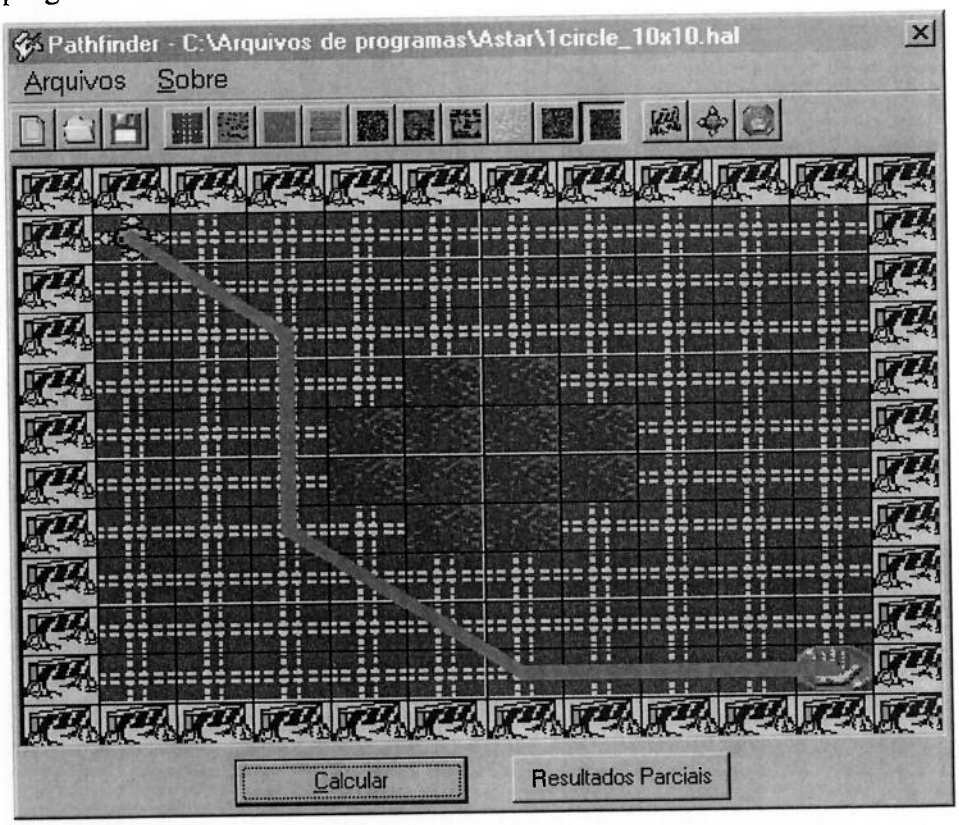

Fig. 9 . Tela de ambiente resolvido, com discretização pequena (dez por dez pontos), para o Exemplo 1: ambiente em si, pontos inicial e final e o caminho menos custoso.

Na Figura 9, pode-se ver a tela de resultados com visão do ambiente: vê-se o botão "Resultados Parciais"; selecionando-se tal botão, chega-se à tela, em que podem ser vistas as informações de interesse ao estudo de eficiência e desempenho do método **A***, mais especificamente do programa **AStar**: no topo vê-se o tempo total de busca; em cada linha vê-se: número do nó no grafo, coordenadas do ponto referente àquele nó e os valores das funções heurísticas de interesse naquele ponto, naquele nó: *g(nó)*, *h(nó)* e *f(nó)*, sendo esta última soma das duas anteriores. Perceba ter o caminho um total de treze nós, incluindo os pontos S e G.

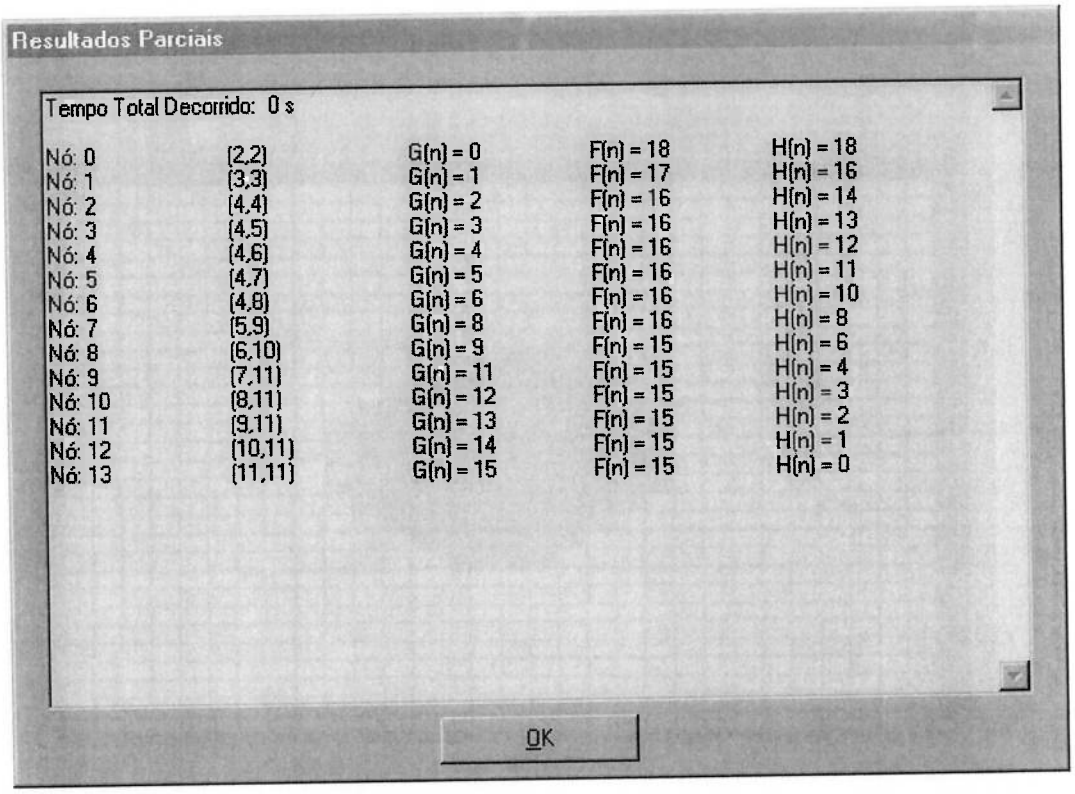

Fig. 10 . Os resultados numéricos do Exemplo 1 processado com discretização pequena.

Após a resolução do Exemplo 1 via **AStar** com discretização pequena, procede-se à solução do mesmo exemplo com maior discretização: a intermediária é utilizada. As duas repostas (gráfica e com detalhamento matemático) podem ser vistas nas Figuras 11 e 12, respectivamente. Perceba ser a resposta assim encontrada diferente daquela achada na discretização pequena.

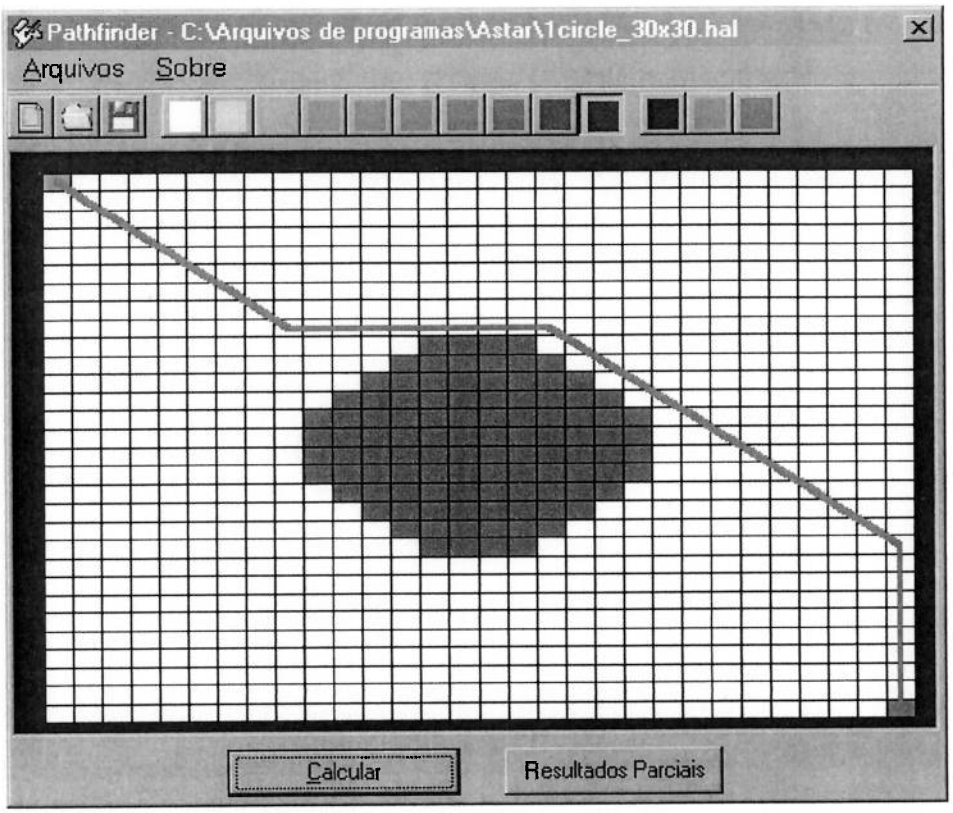

Fig. 11 . O Exemplo 1 com representação adequada à discretização intermediária (trinta por trinta pontos), i.e., com os ícones trocados por cores.

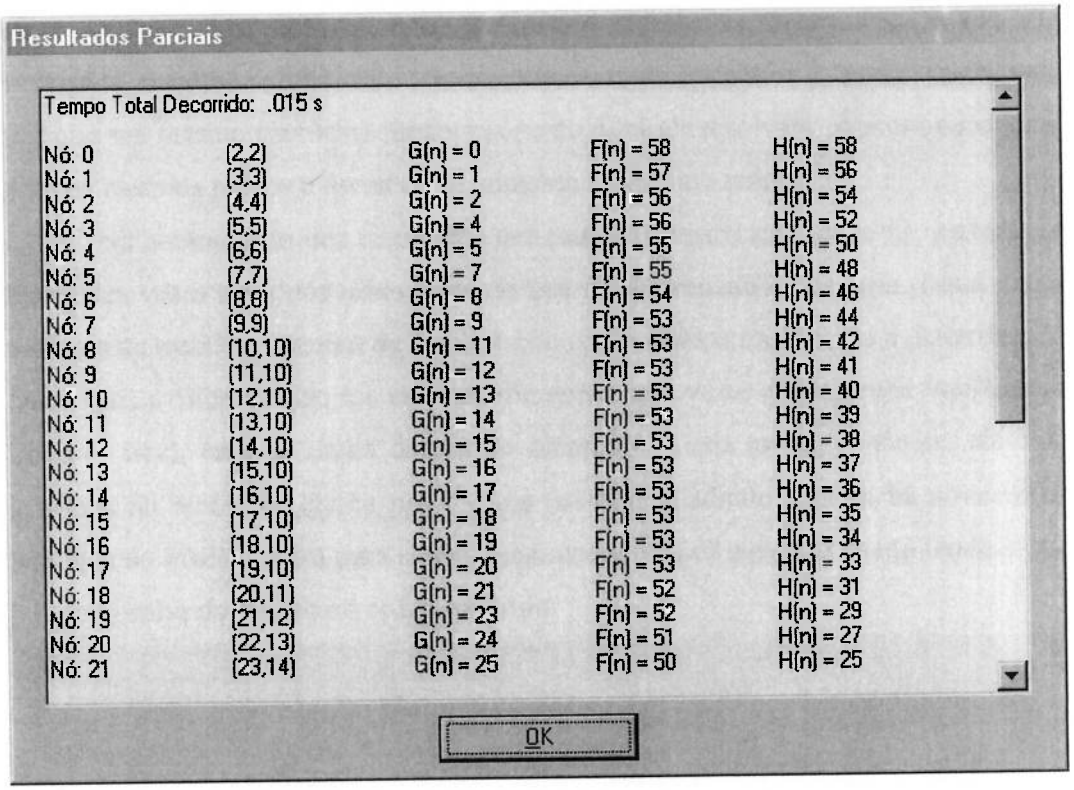

Fig. 12 . Os resultados numéricos do Exemplo 1 processado com discretização intermediária, página um de um total duas.

A respeito da Figura 12, percebe-se que o tempo de busca, pequeno demais para ser medido na discretização pequena, agora é quinze milissegundos. Dado que a discretização no tempo é de um milissegundo, o tempo total de busca foi, naquela discretização, menor que tal valor (um milissegundo). Assim sendo, aquela busca se deu em um tempo pelo menos quinze vezes menor que o da discretização intermediária. Ao longo dos diferentes ambientes e discretizações ficará claro que há distorção na medição dos tempos, dado serem eles baixos demais (i.e., da ordem de grandeza dos tempos de leitura e escrita em disco). Um computador mais lento seria requerido para desenvolver os testes para o **AStar** sem deformações deste tipo. Já os tempos do **SAtest**, bem maiores (quatro ordens de grandeza maiores, da ordem de dezenas de segundos), são bem mensuráveis.

Uma outra observação importante quanto aos tempos de busca: como já exposto, tem o **SA** forte caraterística aleatória. Assim sendo, cada vez que se processa o programa para um dado problema, obtém-se um tempo diferente mas semelhante (e possivelmente uma resposta diferente, ainda que sempre ótima ou pelo

menos quase-ótima). O **A***, por sua vez, é totalmente determinístico (há sim heurística, mas suas regras são determinísticas: nada é aleatório), assim sendo, toda vez que um mesmo problema lhe for proposto, será ele resolvido da mesma maneira, com os mesmos passos e decisões, idealmente no mesmo tempo.

Na continuação dos resultados matemáticos, vistos na Figura 13, percebe-se haver três vezes mais nós nesta resposta que na discretização pequena (trinta e oito ao invés de treze): o número de nós-solução cresceu linearmente com a discretização linear (i.e., a discretização foi, em uma dimensão, três vezes maior, trinta ladrilhos ao invés de dez), estando assim dentro do esperado. Ainda assim, o número total de ladrilhos no ambiente cresce nove vezes (três ao quadrado), assim há novecentos ladrilhos ao invés de cem para serem examinados: é fácil perceber como tende a cair o desempenho do algoritmo como um todo.

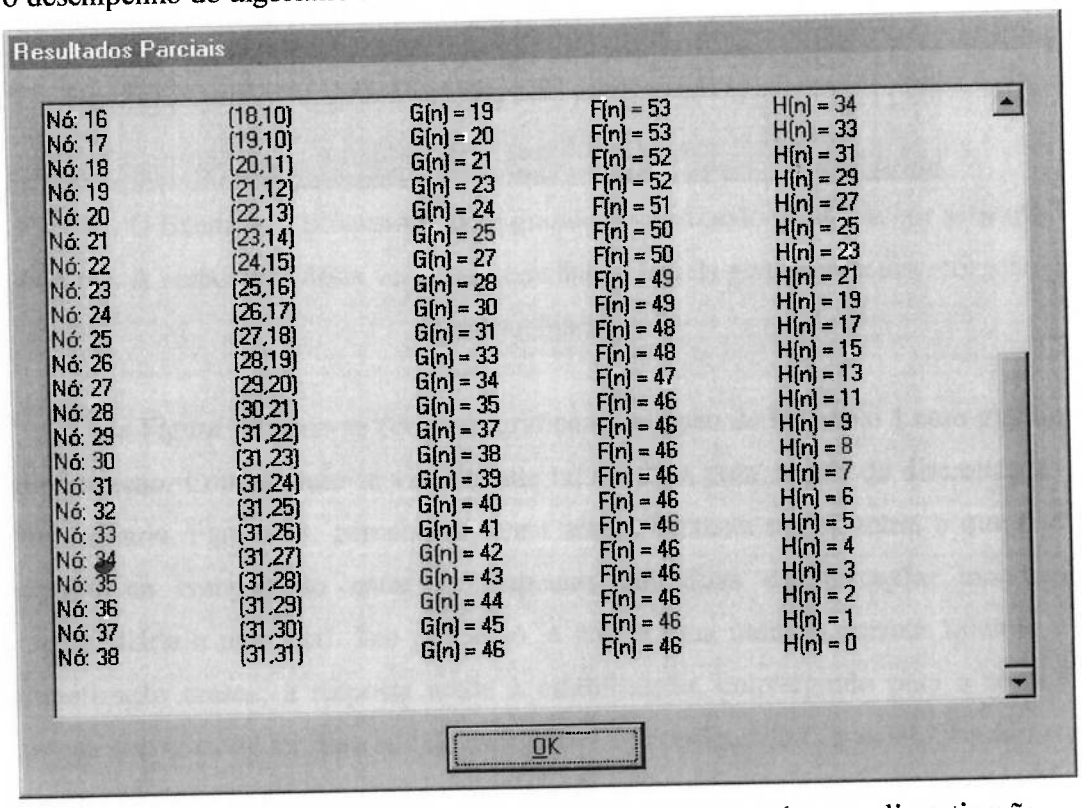

Fig. 13 . Os resultados numéricos do Exemplo 1 processado com discretização intermediária, página dois de um total de duas.

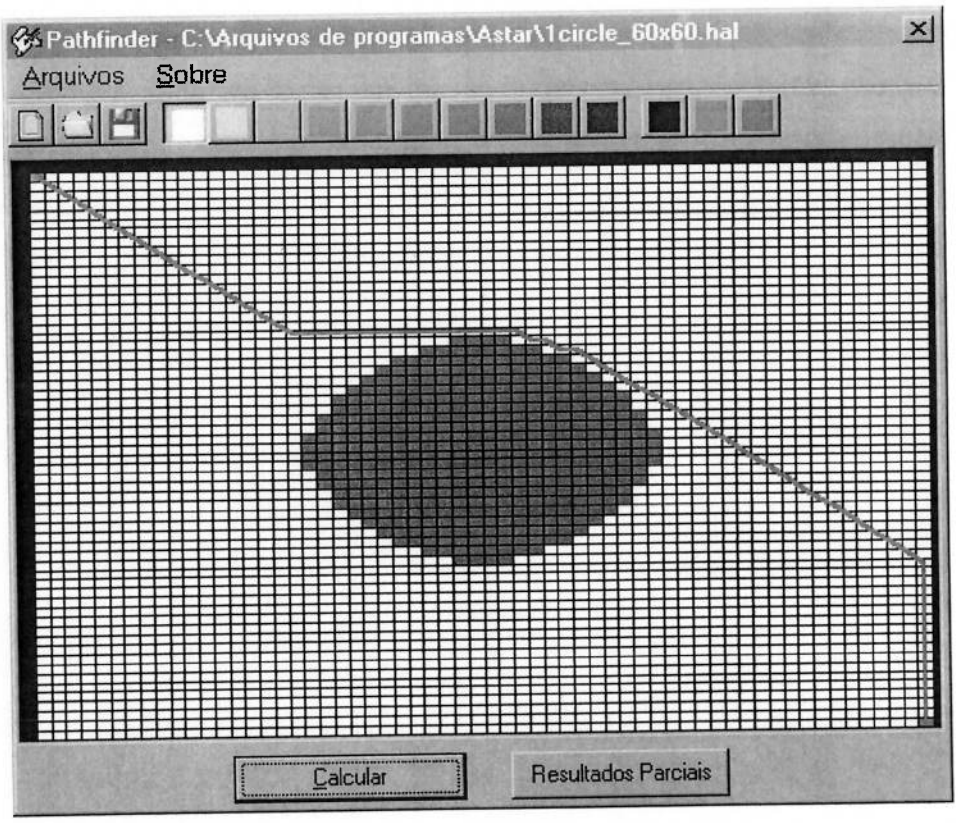

Fig. 14 . O Exemplo 1 processado com grande discretização (sessenta por sessenta pontos). A resposta é, desta vez, bem semelhante àquela gerada pela discretização intermediária.

Na Figura 14 pode-se ver a tela gráfica da solução do Exemplo 1 com grande discretização. Comparando-se visualmente tal resposta com aquela da discretização intermediária, Figura 11, percebe-se serem ambas bastante semelhantes, o que não ocorreu na comparação entre as respostas das duas discretizações menores (intermediária e pequena). Isto por si só já leva a uma indução correta: quando a discretização cresce, a resposta tende à estabilização, convergindo para a menos custosa que se pode ter para aquele dado estilo de configuração espacial. Quando se fala em "estilo" de configuração espacial, refere-se a duas coisas: à forma de cada azulejo em que se divide o ambiente contínuo e ao tamanho relativo entre tais azulejos: no caso do programa **AStar**, usou-se azulejos quadrados, todos de mesmo tamanho. Há também configuração por quadrados, mas com tamanhos diferentes, i.e., discretização maior para terreno mais variável e menor para trechos mais uniformes. Outra configuração espacial possível é aquela que utiliza hexágonos em

lugar dos quadrados; tal discretização foi inspirada nos tradicionais jogos de estratégia militar de mesa, em que se usam hexágonos – e não quadrados - para posicionar as peças. Em tal caso, a árvore gerada na configuração espacial é uma *hex-tree* (no caso dos quadrados, é uma *quad-tree* ou uma *oc-tree*).

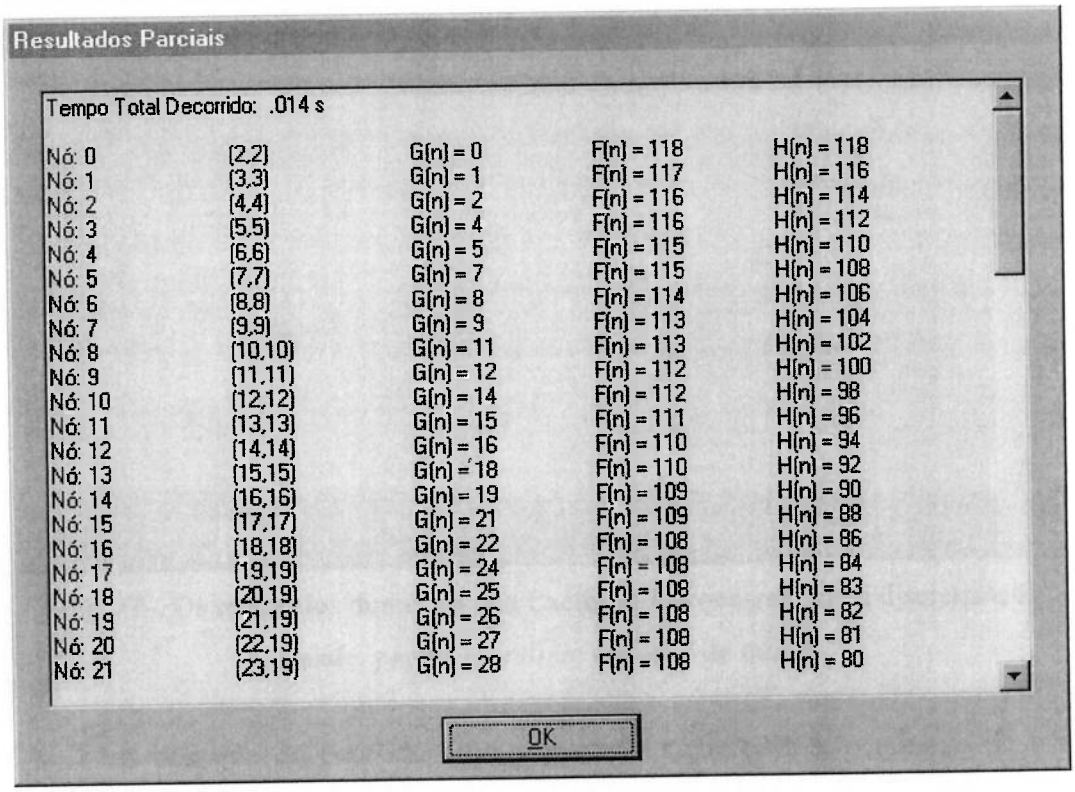

Fig. 15 . Os resultados numéricos do Exemplo 1 processado com discretização grande, página um de um total quatro.

Na Figura 15, vê-se os dados numéricos importantes à análise da solução do Exemplo 1 com discretização grande. Apenas a primeira e a última páginas de resultados (de um total de quatro) são aqui representadas, por seus dados de interesse. São eles: tempo total de busca e número total de nós. Os valores das funções em alguns dos nós - ou pontos - podem também ser vistos nestas duas páginas. A distorção na medição nos tempos se faz presente, a exemplo do que ocorreu na discretização intermediária: aquele problema, mais simples (mesmo ambiente, com menor discretização), foi resolvido em quinze milissegundos, tempo um pouco maior que este, catorze milissegundos; esperava-se, claro, que tivesse o caso mais simples um tempo total de solução menor.

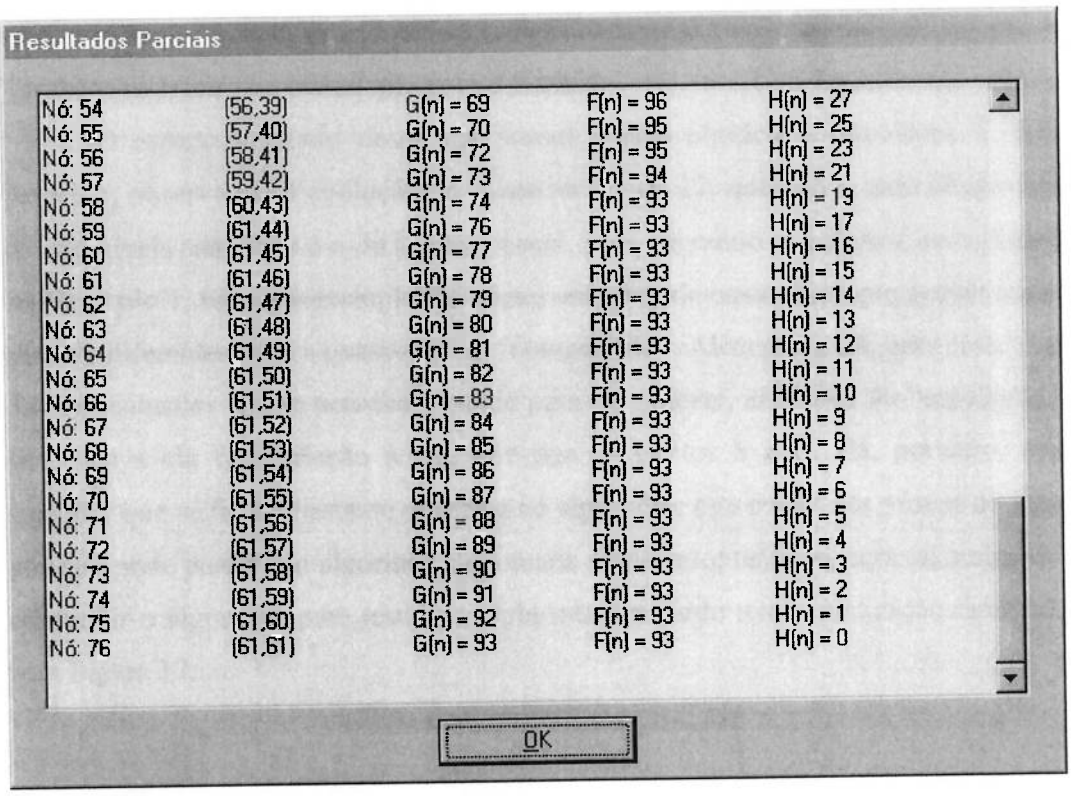

Fig. 16 . Os resultados numéricos do Exemplo 1 processado com discretização grande, página quatro de um total de quatro.

A exemplo do ocorrido na discretização intermediária, o número de nós-solução cresceu linearmente com a discretização linear (i.e., a discretização foi, em uma dimensão, duas vezes maior, sessenta ladrilhos ao invés de trinta), como esperado. Entretanto, o número total de ladrilhos no ambiente cresce quatro vezes (dois ao quadrado), assim há três mil e seiscentos ladrilhos ao invés de novecentos para serem examinados: a conclusão não poderia ser diferente daquela feita na discretização anterior: cai o desempenho do algoritmo com o aumento da discretização, devido ao crescimento quadrático da esperança do número de nós visitados pela busca.

Resumindo, o primeiro processamento deu-se com cem ladrilhos no total, depois novecentos e finalmente três mil e seiscentos ladrilhos.

O Exemplo 2

Como feito no Exemplo 1, o processamento deste exemplo foi realizado com as seguintes discretizações no **AStar**: quadrados de dez, trinta e sessenta "ladrilhos"

ou "azulejos" de lado, como se vê, respectivamente, nas Figuras 18, 20 e 23. Também se processou o exemplo com o **SAtest**.

O espaço utilizado desta vez possui quatro obstáculos circulares. É fácil perceber, observando a evolução da busca na Figura 17, que não é nada óbvio qual dos possíveis caminhos é o de mínimo custo, pois, de modo semelhante ao ocorrido no Exemplo 1, há neste exemplo diferentes soluções de custos bastante semelhantes: são três diferentes custos quase-ótimos "competindo". Além disto, há, para cada uma das três soluções que se percebe olhando para o ambiente, uma solução "espelhada", simétrica a ela com relação à reta que liga os pontos S e G. Há, portanto, seis soluções que se fazem bastante atraentes ao algoritmo: esta é uma das provas de fogo por que pode passar um algoritmo de tomada de decisão: tenta-se, com tal ambiente, confundir o algoritmo para testar sua robustez. Isto tudo tem visualização facilitada pela Figura 17.

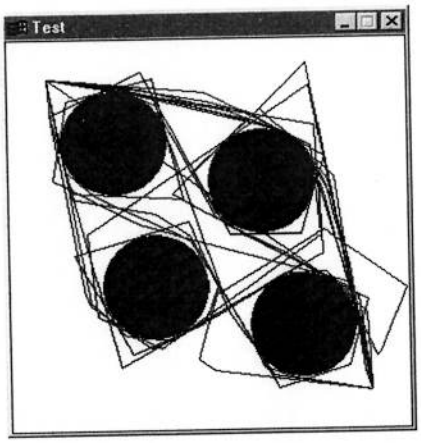

Fig. 17 . O progresso da busca por um caminho pouco custoso via *Simulated annealing*. Representam-se aqui dezesseis passos consecutivos do algoritmo.

Representa-se, na Figura 17, dezesseis passos consecutivos do algoritmo para auxiliar na compreensão da evolução do algoritmo e da qualidade das soluções, pois dá, graficamente, uma boa idéia de como a busca evolui no tempo. Tal representação é obtida da seguinte maneira: quando um novo passo é processado, a mais velha das curvas é apagada (ela é a curva que foi gerada em primeiro lugar dentre as dezesseis). Obtém-se, assim, um "rastro" da busca.

Há algoritmos que, para casos gerais, têm bom desempenho e conseguem gerar soluções próximas das ótimas, mas quando aplicados a ambientes projetados

para serem críticos, acabam confundindo-se, dando soluções ruins e/ou em tempos grandes. Isto pode ocorrer mesmo com algoritmos robustos, que normalmente se comportam bem, como o **SA**: eles podem ser prejudicados, em casos capciosos como este Exemplo 2, por funções de avaliação mal formuladas. Adicionalmente, discretizações incorretas do espaço podem ocorrer no caso de algoritmos como o **A***, corroborando para o mesmo fenômeno; é claro que o **SA** não corre este risco. Este exemplo mostrará como exatamente este ponto citado é crucial: sucederá o **SA** onde falhou o **A***, devido a algumas das configurações espaciais terem sido inadequadas por terem tido detalhamento insuficientemente.

Ao contrário do que se fez no Exemplo 1, a resolução via **AStar** será mostrada antes daquela via **SAtest**.

Na Figura 18 pode-se ver a solução com a pequena discretização para o Exemplo 2. Nota-se que o robô desviou de todos os obstáculos, evitando caminhos que tivessem qualquer constrição, pois pela baixa discretização ficaram eles muito custosos (aos olhos do programa), o que não é verdade, como mostrarão claramente as respostas dadas pelo **SAtest** e pelas duas maiores discretizações do **AStar**.

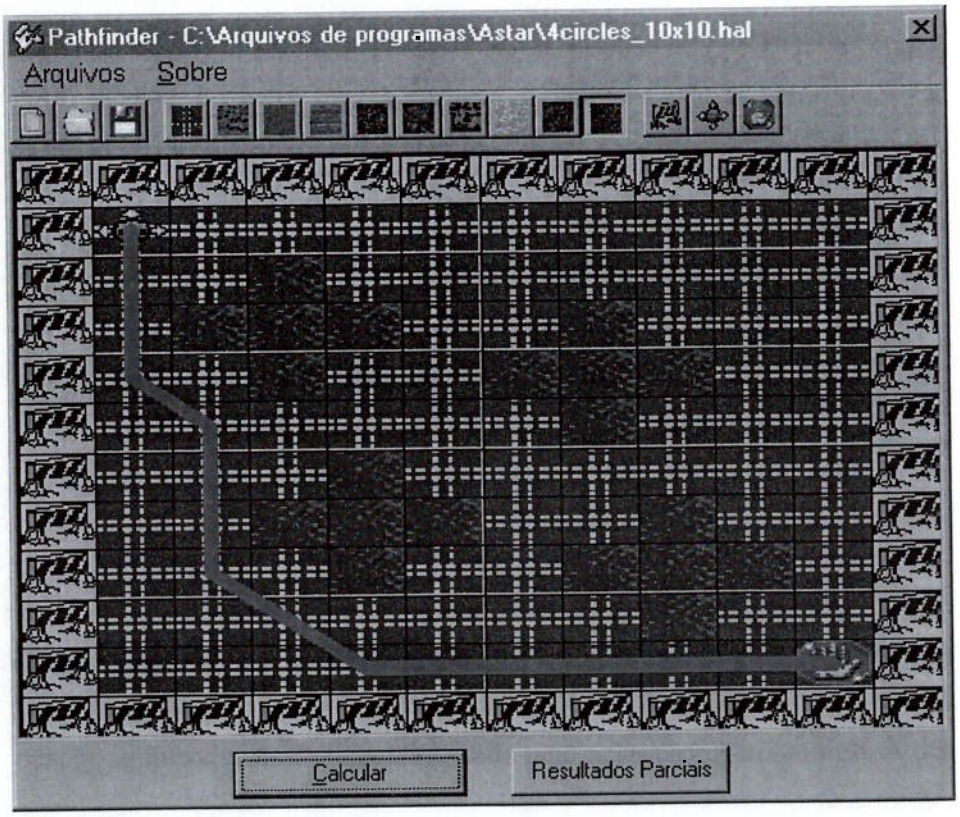

Fig. 18 . O Exemplo 2: ambiente, pontos inicial e final e o caminho menos custoso na discretização pequena (dez por dez pontos). Perceba que o robô desviou de todos os obstáculos.

A Figura 19 mostra ter o caminho um total de quinze nós, incluindo os pontos *S* e *G*. Para efeito de comparação, lembrar que no Exemplo 1, de ambiente mais simples, a resposta teve treze nós. De modo similar ao que ocorreu naquele exemplo, o tempo total de busca foi pequeno demais para ser computado, i.e., menor que um milissegundo.

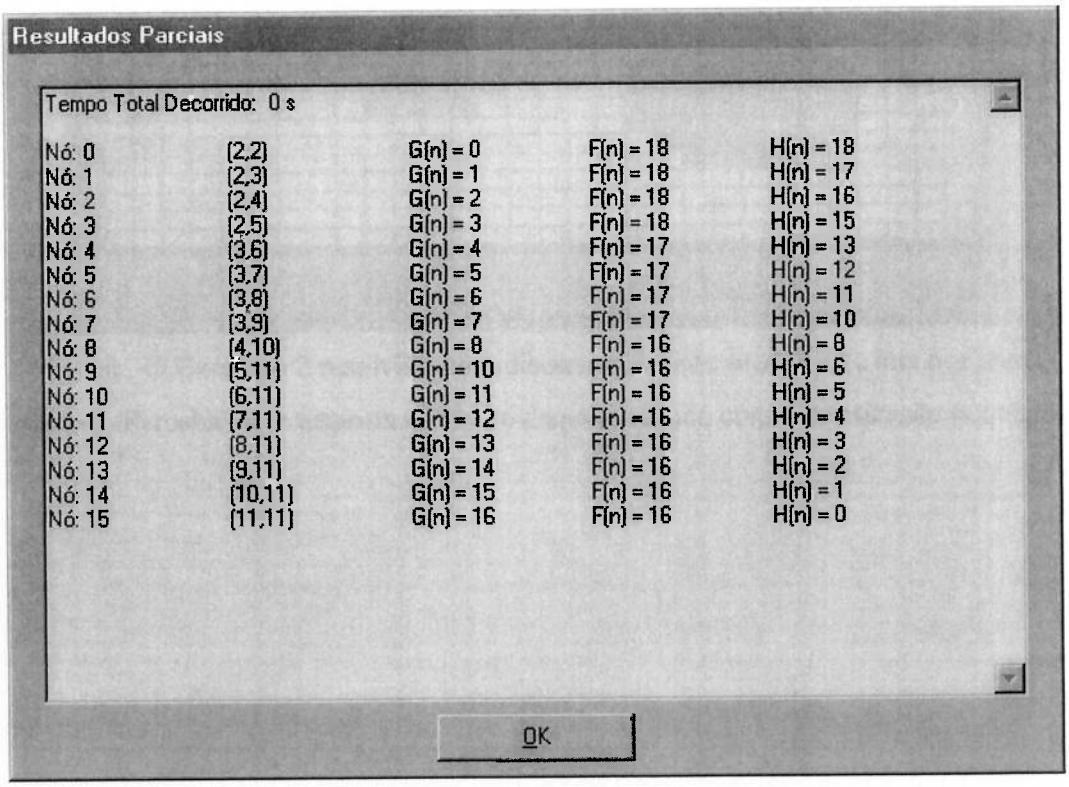

Fig. 19 . Os resultados numéricos do Exemplo 2 processado com discretização pequena.

Na Figura 20 pode-se perceber que a resposta é bem diferente daquela achada com a utilização da discretização pequena, exatamente como ocorreu no Exemplo 1. Dos três tipos de resposta possíveis (refere-se aqui às de pesos semelhantes, já citadas na introdução do exemplo), cada uma delas escolheu um tipo. A discretização grande fará com que o algoritmo escolha o terceiro tipo, como se verá adiante (Figura 23).

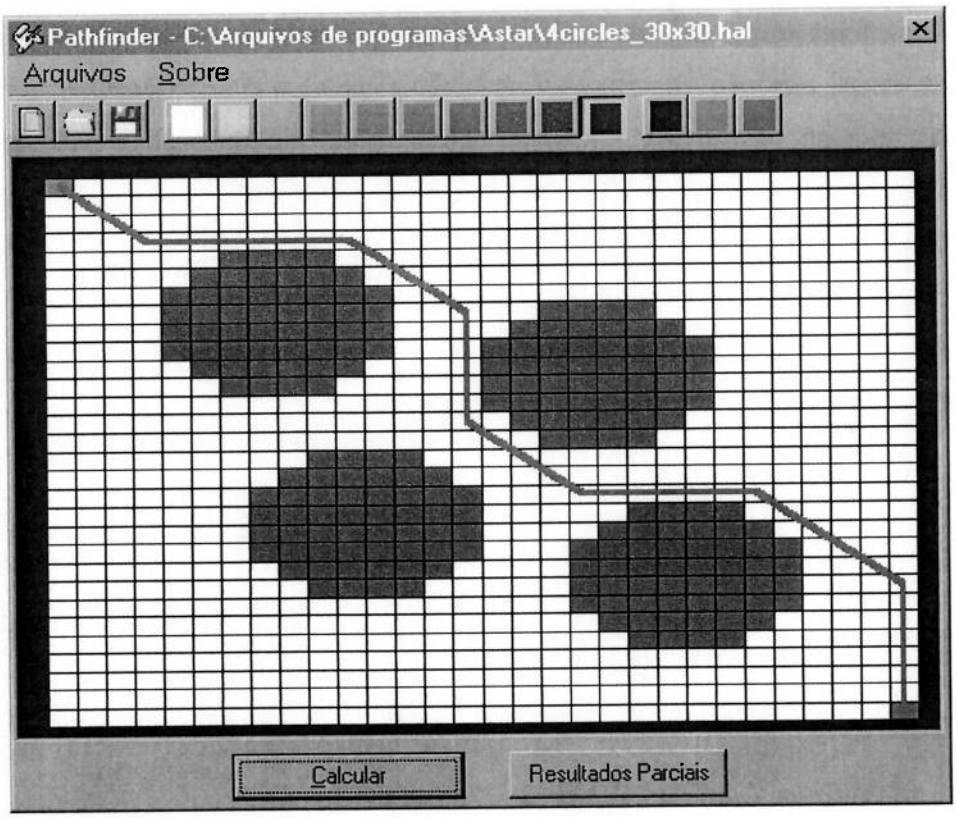

Fig. 20 . O Exemplo 2 resolvido com discretização intermediária (trinta por trinta pontos). Perceba ser a resposta diferente daquela achada com discretização pequena.

Na Figura 21, tem-se os resultados numéricos referentes à Figura 20. O tempo, pequeno demais para ser medido na discretização pequena, agora é cinco milissegundos nesta discretização, a intermediária. A distorção na medição nos tempos aparece, a exemplo do exposto no Exemplo 1: fazendo-se uma comparação com aquele exemplo, quando resolvido com discretização intermediária, percebe-se que aquele problema, mais simples (a mesma discretização para o ambiente menos complexo do Exemplo 1), foi resolvido em quinze milissegundos, tempo três vezes maior que este, quando se esperava que este processamento fosse tomar um tempo maior que aquele.

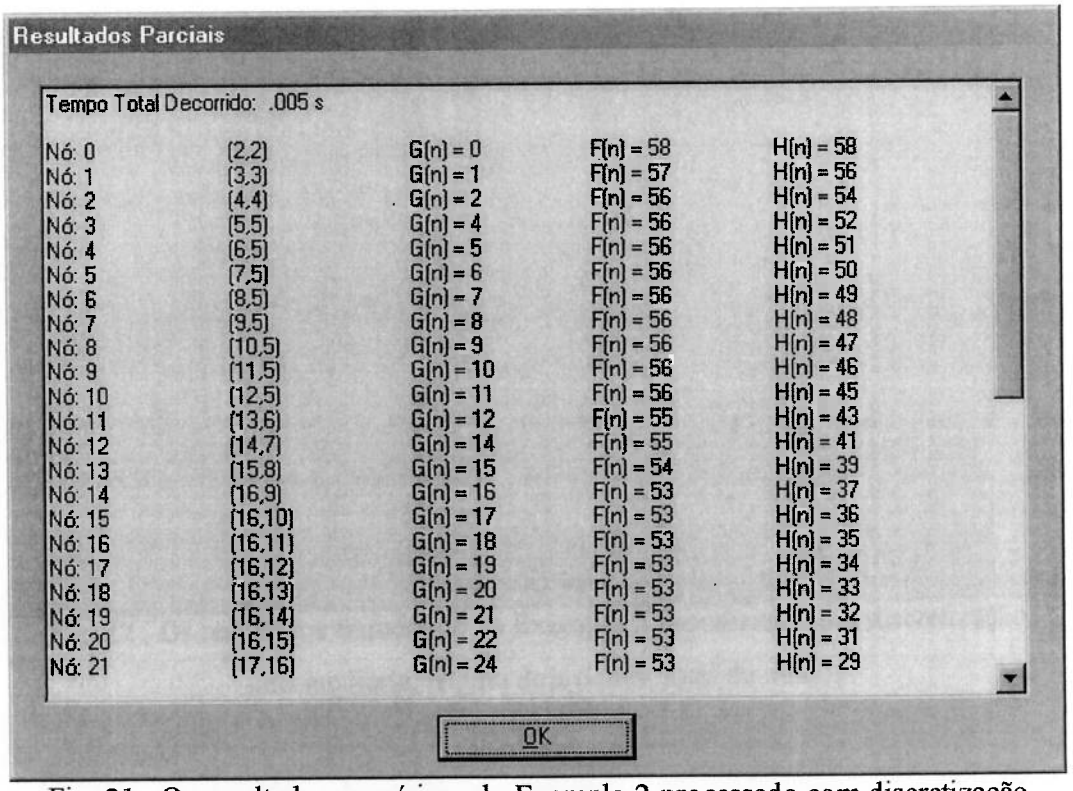

Fig. 21 . Os resultados numéricos do Exemplo 2 processado com discretização intermediária, página um de um total duas.

Percebe-se na Figura 22 que há aproximadamente três vezes mais nós nesta resposta que na discretização pequena para o mesmo exemplo (quarenta e dois ao invés de quinze). Posto que a discretização foi, em uma dimensão, três vezes maior (o quadrado que representa o ambiente tendo trinta ao invés de dez ladrilhos de lado), verifica-se novamente que o número de nós-solução cresceu linearmente com a discretização linear.

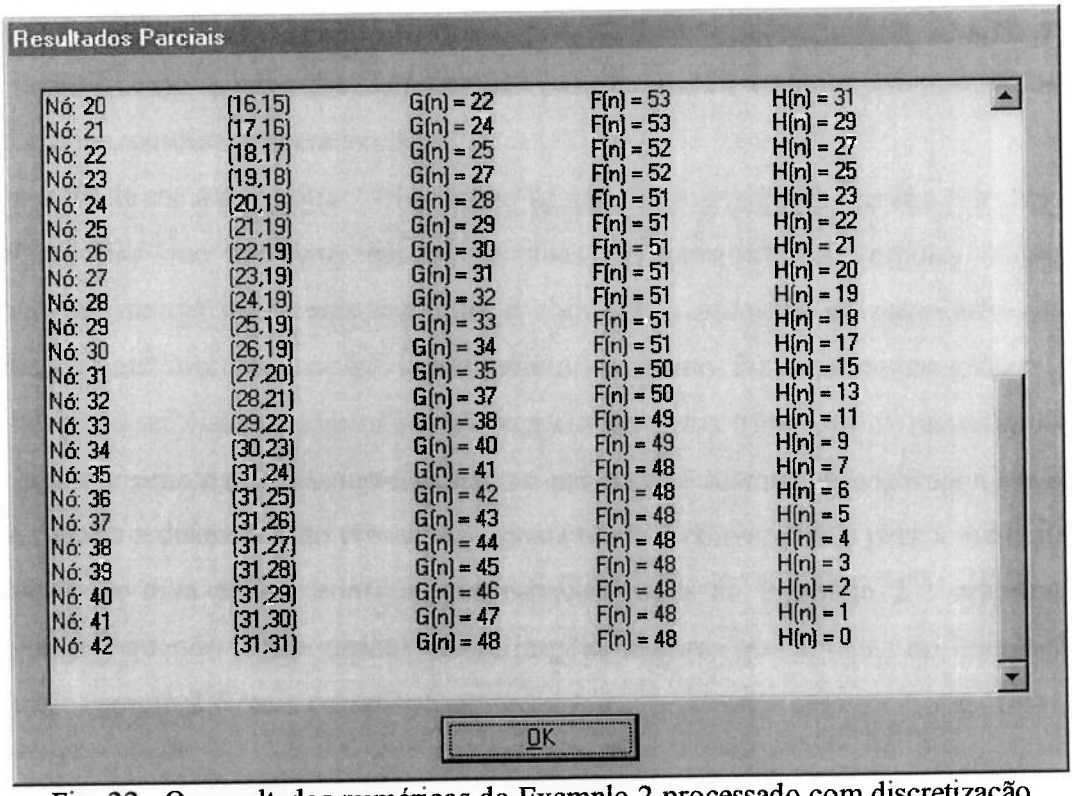

Fig. 22 . Os resultados numéricos do Exemplo 2 processado com discretização intermediária, página dois de um total de duas.

Na Figura 23 tem-se o Exemplo 2 processado com grande discretização. A resposta é, desta vez, diferente daquela encontrada pela discretização intermediária (enquanto que no Exemplo 1 as duas melhores discretizações geravam respostas muito semelhantes) e também daquela gerada com a pequena, o que é de bastante interesse didático: há três tipos diferentes de solução para este caso, como já citado: o que ocorreu aqui é que cada discretização concluiu ser melhor uma delas; isto por si só mostra como o Exemplo 2, projetado para confundir os dois programas, ambos algoritmos, confunde apenas o **A***.

Pode vir a pergunta: "Mas como se sabe que a resposta gerada pelo **SA** é a melhor? Por que não uma das outras duas em questão?". A resposta é que se processou muitas vezes este exemplo, e com vários números de segmentos para a trajetória; até mesmo usou-se várias trajetórias, como faz usualmente a literatura. Como pode ser visto na Figura 17, a busca alterna pelos três pares de respostas antes de escolher uma das duas pertencentes ao par menos custoso. Ainda vale a idéia de que quando a discretização cresce, a resposta tende à convergência para a melhor que se pode ter para aquela configuração espacial. Aqui no Exemplo 2 a resposta só convergirá quando forem usadas discretizações maiores que aquelas do Exemplo 1, pois o Exemplo 2 é mais complexo.

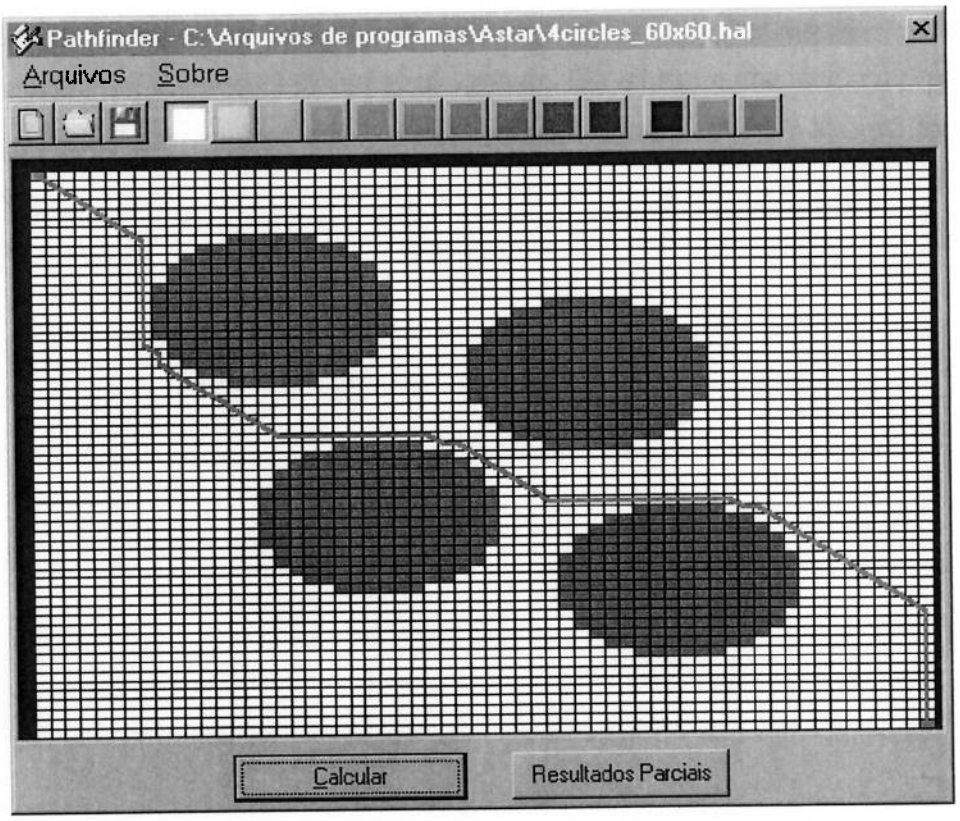

Fig. 23 . O Exemplo 2 processado com grande discretização (sessenta por sessenta pontos). A resposta é diferente da encontrada pela discretização intermediária.

Na Figura 24 tem-se a primeira página de resultados numéricos do Exemplo 2 processado via **AStar** com discretização grande. Ela mostra que o tempo de busca é treze milissegundos, maior que o da discretização intermediária para este mesmo exemplo, como esperado, mas menor que o tempo na mesma discretização no Exemplo 1, cujo ambiente é mais simples: esta é mais uma distorção devida aos baixos tempos totais de solução.

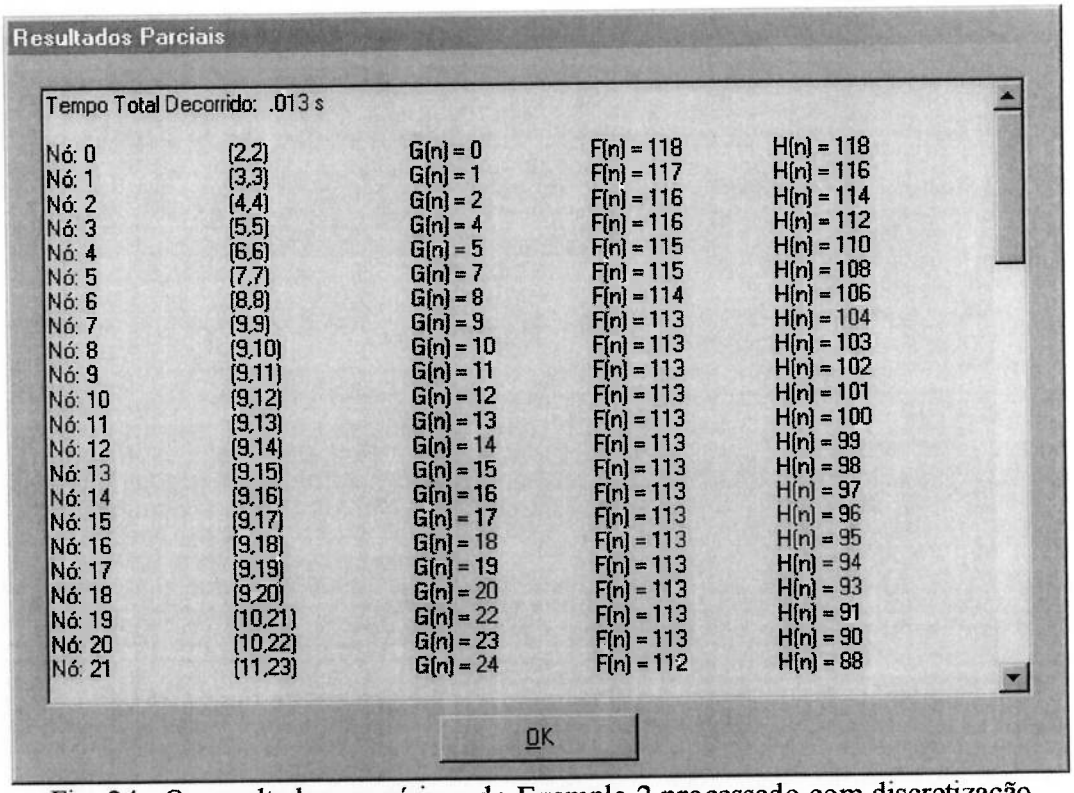

Fig. 24 . Os resultados numéricos do Exemplo 2 processado com discretização grande, página um de um total quatro.

Na Figura 25, tem-se a página final da resposta em formato numérico (como feito no Exemplo 1, apenas a primeira e a última páginas são aqui representadas, por seus dados de interesse). Perceba que a discretização foi, em uma dimensão, duas vezes maior que na discretização intermediária (sessenta ao invés de trinta ladrilhos), e há duas vezes mais nós nesta resposta que naquela (oitenta e três nós ao invés quarenta e dois): novamente o número de nós-solução cresceu linearmente com a discretização linear, como esperado.

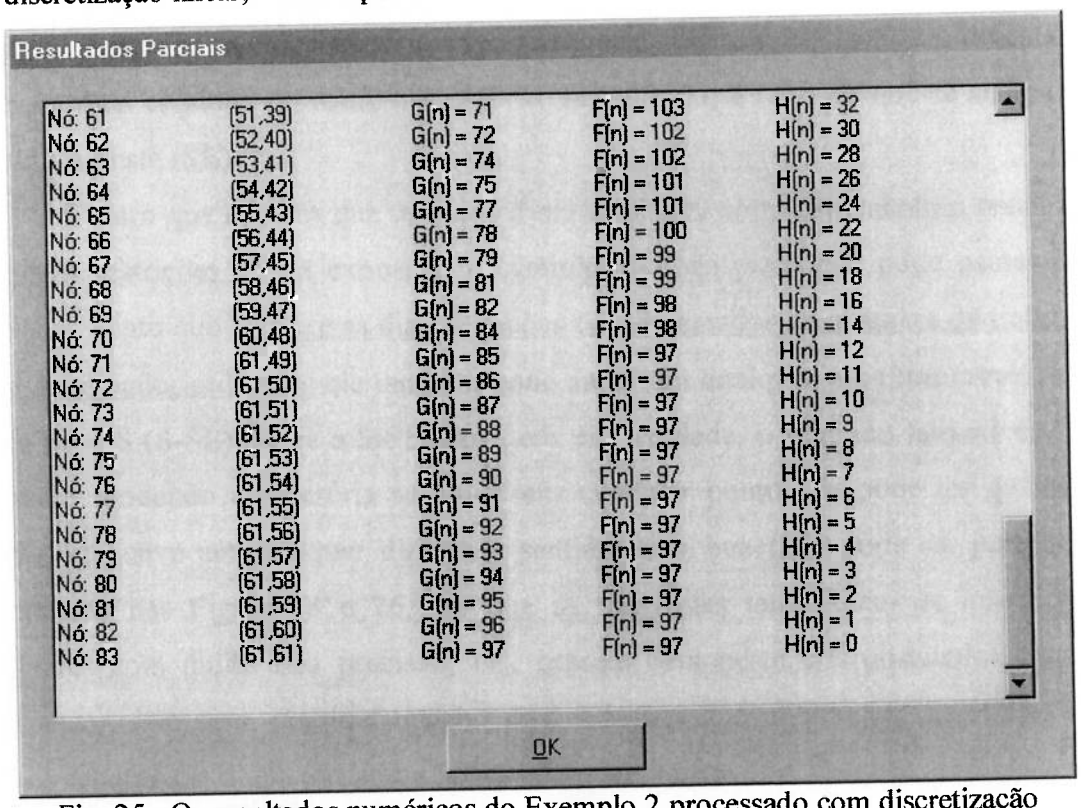

Fig. 25 . Os resultados numéricos do Exemplo 2 processado com discretização grande, página quatro de um total de quatro.

Na Figura 26 vê-se a resolução do Exemplo 2 pelo programa **SAtest**. Percebe-se, clara e intuitivamente, como é maior a qualidade de tal resposta em comparação com as três geradas com a combinação de configuração espacial e busca em grafos, programa **AStar** (Figuras 18, 20 e 23). Isto ocorre por dois motivos: o primeiro é que o espaço discretizado pela configuração espacial obriga o caminho a passar pelos centros dos ladrilhos. O segundo motivo é que deve o caminho seguir uma das quatro direções básicas de movimentação: vertical, horizontal e diagonais. Boa parte do caminho do robô, nas Figuras 18, 20 e 23, é feito no que se chama de sentido sudeste (SE).

É claro que a busca que utiliza o *Simulated annealing* não encontra nenhuma das duas restrições acima expostas: o caminho por ela planejado pode passar por qualquer ponto que respeite as discretizações do processador da máquina de trabalho e do compilador utilizado; ele também pode andar em qualquer proa (ou rumo), i.e., entre SE e S (S-SE), entre o S-SE e o S etc. Em verdade, o segundo fato advém do primeiro: podendo a trajetória se situar em qualquer ponto, ela pode ter qualquer forma, inclusive em qualquer direção e sentido. Este benefício pode ser percebido claramente nas Figuras 8 e 26, em que as trajetórias tangenciam os obstáculos circulares com muito boa precisão, i.e., passam bem perto dos obstáculos sem o interceptar. Isto seria possível mesmo com soluções que utilizassem trajetórias de menos segmentos, o que tornaria a busca bem mais rápida.

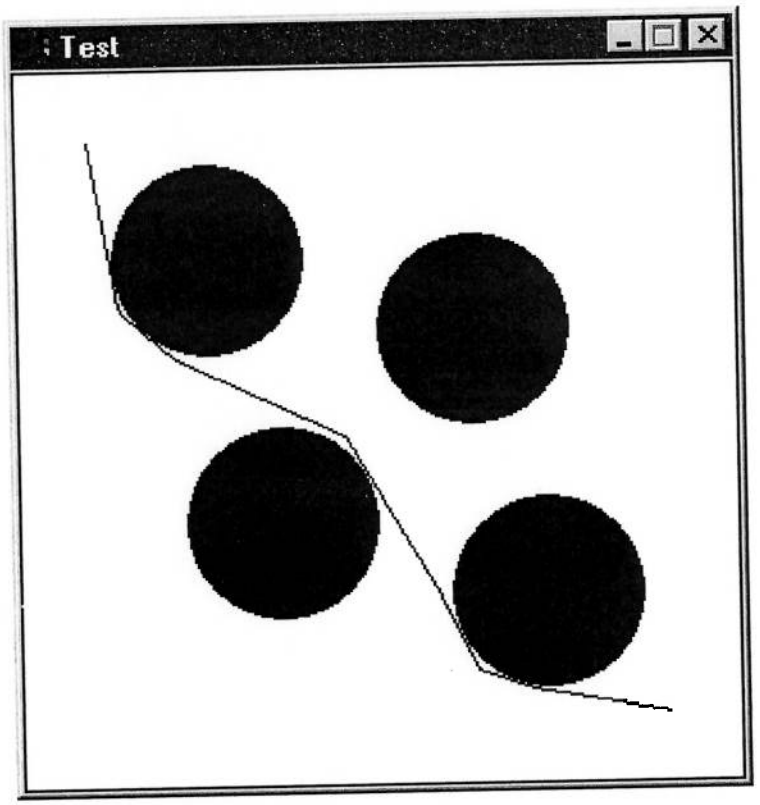

Fig. 26 . Resolução do Exemplo 2 pelo programa **SAtest**. Percebe-se ser maior a qualidade desta resposta em comparação com as geradas pelo programa **AStar**.

Nas respostas geradas pelo **AStar**, percebe-se que a busca não tangencia o obstáculo, mas sim passa a *L/2* na horizontal e na vertical e a $(L/2)*2^{1/2}$ na diagonal (*L* é a dimensão do lado dos azulejos). Isto pode parecer pouco, mas em ambientes com caminhos estreitos por onde pode o robô passar (i.e., suas dimensões são pouco maiores que as da passagem), o **SAtest** fará com que robô procure outro caminho, de custo não-mínimo, pensando ser o "túnel" estreito demais: isto ocorrerá quando for sua largura menor que *L* no melhor caso da configuração – ou mesmo se menor que o dobro de *L*, no pior caso. Se não houver caminho alternativo, ele dirá que não há caminho ligando os pontos *S* e *G*, o que não é verdade.

Já no caso do **SA**, se houver espaço para o robô passar (leia-se a parte mais estreita do "túnel" tiver a dimensão do robô, respeitadas as discretizações de processador e compilador, como já citado), ele usará tal caminho.

Percebe-se, com isto e com a observação da Figuras 17 e 26, ser o **SA** realmente robusto, pois consegue "enxergar" a pequeníssima diferença de custo entre

todas as soluções, convergindo, assim, para uma das duas soluções de menor custo. Tais soluções são aquelas em que o robô tangencia três dos obstáculos: "por fora", daí "por dentro" e novamente "por fora" (vide Figura 26).

Uma observação relativa aos dois exemplos acima: comparando-os entre si, percebe-se que no primeiro caso se obteve treze, trinta e oito e setenta e seis nós, respectivamente, para as discretizações pequena, intermediária e grande. No Exemplo 2, quinze, quarenta e dois e oitenta e três. Treze é muito próximo de quinze; de modo similar, trinta e oito é muito próximo de quarenta e dois e setenta e seis é muito próximo de oitenta e três. Percebe-se que os números de nós calculados para ambos os exemplos são muito semelhantes, quando se compara dois resultados de mesma discretização. Isto se dá por causa da restrição de direções em que o robô pode se locomover no "tabuleiro de xadrez" em que busca o **A***. O **SA**, com sua maior liberdade, encontra respostas de maior qualidade, possibilitando melhor visualização das diferenças entre a resposta do Exemplo1 e a do Exemplo 2 (vide Figuras 8 e 26).

O Exemplo 3

Como exposto durante a apresentação das novidades deste trabalho, há uma possível variação do **SA** básico que é trabalhar com várias soluções dependentes entre si ao mesmo tempo. Em tal implementação, os dados obtidos de todas elas são utilizados para calcular uma variância que é um estimador de quão bem comportada está a convergência do algoritmo, tudo isto para controlar a taxa de resfriamento α a fim de se ter uma convergência tão acelerada quanto possível de modo que ainda se garanta a boa qualidade da solução. Tal procedimento foi implementado, com 16 trajetórias simultâneas de 10 segmentos cada; a tela gráfica de sua solução é exposta na Figura 27. Perceba que o problema é o mesmo do Exemplo 2.

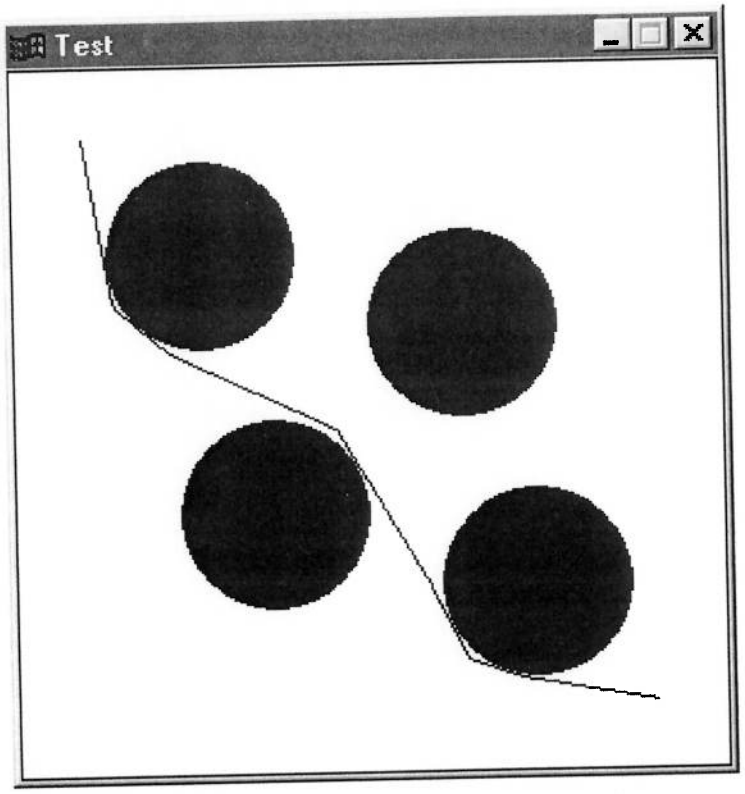

Fig. 27 . Resolução do Exemplo 3 pelo programa **SAtest**: a melhor solução gerada pela aplicação do *Simulated annealing* com múltiplas trajetórias simultâneas.

É interessante comparar as soluções que o programa **SAtest** achou para este exemplo e o anterior, Exemplo 2 (Figuras 26 e 27, respectivamente). Nota-se, com clareza, a semelhança visual entre ambas: são, sem dúvida, eficazes os dois métodos, i.e., atingem o resultado desejado, soluções no mínimo quase-ótimas. Sendo ambas de qualidade semelhante, passa-se à análise de desempenho, i.e., eficiência de ambas: o método tradicional da literatura, de múltiplas trajetórias, tomou um tempo médio de busca de aproximadamente 28 segundos, enquanto que o método por este trabalho proposto levou em torno de 25 segundos em média em sua busca. Percebe-se, assim, terem os métodos tempos semelhantes – rodado o mesmo exemplo muitas vezes -, sendo que o método proposto neste trabalho tem um resultado ligeiramente melhor como um todo: a melhor combinação eficiência e eficácia, ou seja, desempenho do algoritmo e qualidade da solução, respectivamente.

O Exemplo 4

Este caso é *sui generis*, por ser um labirinto, e com paredes apenas verticais e horizontais, portanto de facílima decomposição espacial: fica, assim, potencialmente, a solução pelo **A*** muito boa, pois não existem as inexatidões inerentes à configuração espacial. Isto já foi exposto quando se citou no item 4.4 os casos em que é interessante se aplicar configuração espacial com busca em grafos.

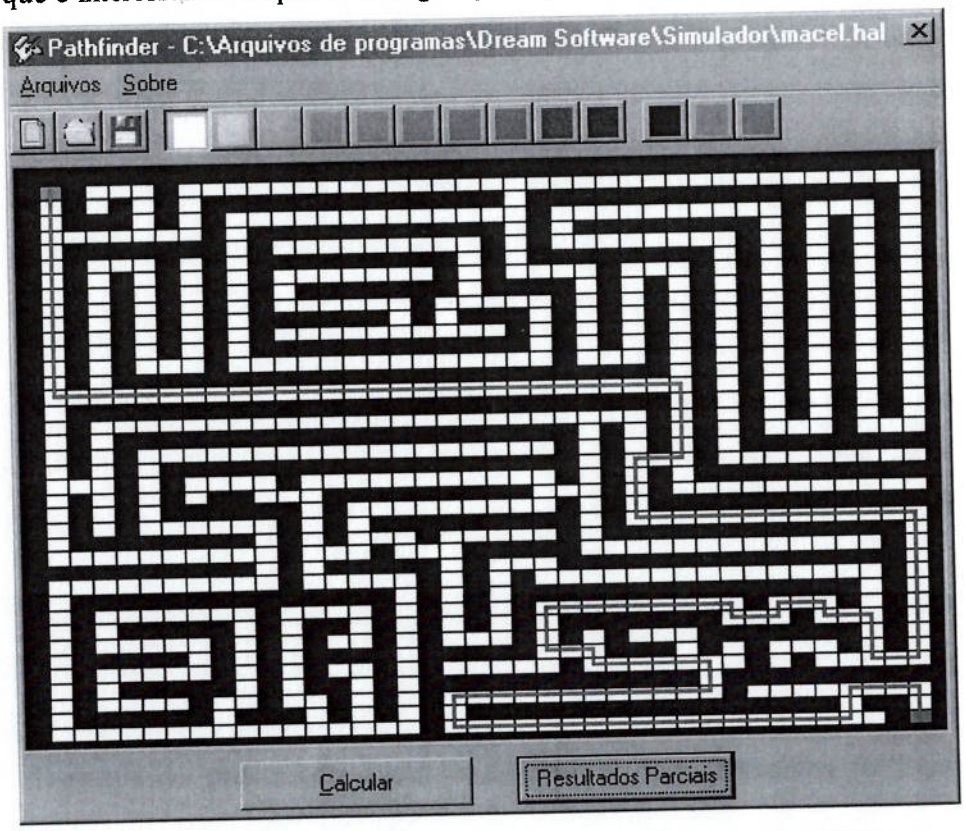

Fig. 28 . Resolução do Exemplo 4 pelo programa **AStar**.

Como esperado, o **A*** comportou-se bem: demorou 50 milissegundos para achar solução deste caso tão específico, um labirinto de terreno binário (só há, no cenário, pesos um e infinito).

O SA, por sua vez, deve se comportar bem neste caso, claro, mas sua aplicação é desnecessária, pois geraria a mesma solução que o **A***, mas em tempo significativamente maior (três, quatro ordens de grandeza). Sendo assim, ele nem será aplicado a este exemplo, que foi projetado exatamente para mostrar que o *Simulated annealing*, mesmo com todos seus prós, pode ser um enfoque superdimensionado, dependendo do caso.

6.3 Trajetórias excessivamente complexas

Um aumento na ordem da trajetória a ser planejada pode melhorar a qualidade da resposta (e a suavidade da trajetória em questão), mas há risco de que situações adversas ocorram. Assim sendo, deve-se ter cuidado ao buscar tais melhorias.

Serão agora expostos os dois fenômenos indesejáveis que podem ocorrer com o aumento da complexidade da trajetória.

Agrupamentos de Vértices

O primeiro risco que se corre quando se aumenta o número de segmentos da trajetória poligonal é o da formação de aglomerações ou agrupamentos (um grupo de vértices, todos próximos entre si).

Os agrupamentos de pontos podem diminuir muito a velocidade de convergência do processo. Para que haja redução de custo de uma trajetória em uma iteração, tal grupo de vértices precisa se mover inteiro para a mesma direção. Supondo que a probabilidade de um vértice se mover numa direção de redução de custo seja cerca de ½, a probabilidade de m vértices se moverem na mesma direção seria em torno de $½^m$ (m é o número de vértices no agrupamento, sendo um natural positivo geralmente maior que três). Como a variação da trajetória fica menos freqüente (pois se tem uma taxa muita alta de rejeição de soluções candidatas), o processo de recozimento precisa ter sua velocidade diminuída também. A Figura 29 mostra o ambiente do Exemplo 1, processado agora com trinta e dois segmentos (o Exemplo 1 foi solucionado com apenas nove segmentos).

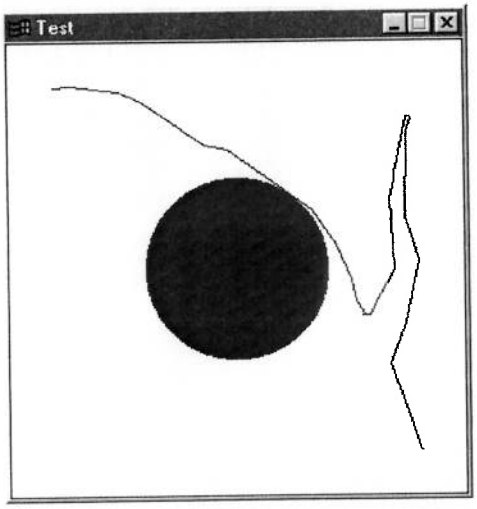

Fig. 29 . O *Simulated annealing* aplicado com vértices demais.

Pode-se claramente observar que o processo está no momento traçando uma trajetória que nem mesmo é próxima de um mínimo local não-global.

Perceba que há um trecho, um "espinho" inteiro que, se retirado da trajetória, levaria a busca a uma solução bem próxima da ótima. Isto significa exatamente retirar o excesso de vértices. Assim, quando muitos vértices permanecem próximos entre si após um número pré-fixado de iterações, pode-se convertê-los todos em um único vértice, reduzindo assim o agrupamento. É claro que o vetor que descreve a trajetória será reduzido de acordo, o que também contribui para o desempenho.

Dando Voltas Desnecessárias

Usar ordens excessivamente altas para as trajetórias traz um outro risco sério: ao se adicionar segmentos desnecessários à trajetória poligonal, pode-se permitir que sejam tomados novos caminhos (de custo não-mínimo, em volta dos obstáculos, portanto indesejáveis). Um exemplo disto se vê na Figura 30, em que se resolve o ambiente do Exemplo 1 com uma trajetória com dezesseis segmentos (o Exemplo 1 foi solucionado com apenas nove segmentos).

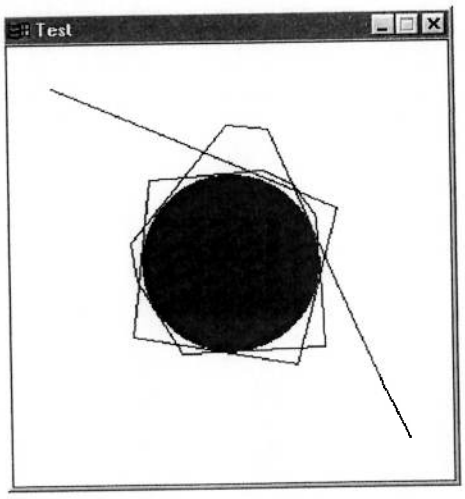

Fig. 30 . Outro perigo do incremento excessivo na complexidade da trajetória.

A situação vista na Figura 30 é de rara ocorrência quando se utiliza uma trajetória poligonal de menos de dez segmentos, mas fica cada vez mais freqüente com o aumento do número de segmentos.

Evitando os Riscos em Trajetórias Complexas

Quão desejável é evitar os problemas expostos acima, devidos ao excesso de complexidade da trajetória? Isto pode ser atestado na Figura 8, que mostra a solução gráfica do Exemplo 1. Compare-se visualmente a qualidade daquela solução com a das obtidas com vértices demais, vistas nas Figuras 29 e 30. É fácil perceber a diferença entre se usar o mesmo algoritmo para o mesmo problema, mas com diferentes números de segmentos (nove para o Exemplo 1, dezesseis e trinta e dois para os exemplos acima, Figuras 29 e 30, respectivamente).

Uma observação bastante importante advinda dos vários testes feitos é que, como previsto teoricamente no item 5.3, quando apenas um vértice é alterado por vez, os efeitos indesejáveis de agrupamento (não necessariamente o agrupamento em si) são reduzidos.

6.4 O Parâmetro Lado

O parâmetro lado é mantido constante durante todo o processo, i.e., a área de localização potencial do ponto candidato permanece um quadrado de mesmas dimensões, *lado x lado*.

Houve uma tentativa de reduzi-lo com a redução da temperatura, mas o sistema perdia, algumas vezes, sua característica desejável de sair de "armadilhas". Assim sendo, ficava preso dentro de "poços de custo" (mínimos locais não-globais). Sendo pequeno o lado, a "escada" - que o algoritmo tem para sair de poços de custo - era curta demais para conseguir lhe tirar de tal poço. Após algumas iterações sem conseguir sair e, claro, com a solução não sendo melhorada, o critério de parada concluiria que o estado frio já haveria sido alcançado, e o processo daria como solução aquele mínimo local não-global, o que é extremamente indesejável. A escada de que se fala é exatamente a característica de robustez do **SA**, i.e., sua possibilidade de ir, por algumas iterações, para soluções piores a fim de escapar de mínimos locais não-globais. Se o poço de custo for muito "profundo" e o lado do quadrado de possibilidades para a nova solução candidata for pequeno, serão necessárias muitas soluções piores – que as anteriores – aceitas para que o algoritmo escape dele.

7. CONCLUSÕES

7.1 Introdução às Conclusões

Os objetivos deste trabalho eram, além de estudar o *Simulated Annealing* e entender seus pontos contra e a favor (e, assim, saber quando sua aplicação se justifica), verificar, especificamente, sua aplicabilidade no planejamento de trajetórias com mínimo custo e sem colisões para um robô dentro de um espaço de trabalho bidimensional.

Para viabilizar tal proposta, utilizou-se de exemplos diversos. A idéia era atestar seu sucesso – ou não – na solução de cada um deles. Assim, o sucesso do método poderia ser parcial. Por exemplo, poderia haver eficiência mas não eficácia ou vice-versa, ou mesmo ser o **SA** indicado para alguns tipos de problema de planejamento de trajetória e outros não).

Nesta seção discute-se se tais metas foram alcançadas.

7.2 Conclusões e Comentários Finais

O **SA**, como exposto teoricamente e comprovado com os testes, não tem um grande crescimento em seu tempo médio de busca com o crescimento do ambiente e/ou do número de obstáculos. Assim, ele tem desempenho atraente para problemas complexos.

Comparando as discretizações do **AStar** que se denominou neste trabalho média e grande, percebe-se que a discretização foi, em uma dimensão, duas vezes maior (espaços quadrados com sessenta ao invés de trinta ladrilhos de lado), o que faz com que haja quatro vezes mais ladrilhos para serem examinados: três mil e seiscentos ao invés de novecentos. Seu tempo de processamento, entretanto, não cresce quatro vezes, pois a busca em árvore feita pelo **A*** tem boa eficiência, mas ainda assim cresce mais rapidamente que a busca feita pelo **SA**. Sobre a boa eficiência do **A***: ele é o mais rápido algoritmo de busca que acha caminhos de custo mínimo entre dois nós de um grafo qualquer (Nilsson, 1973).

O que se expõe acima é que mesmo sendo o eficaz e eficiente **A*** a fazer a busca no grafo que representa o ambiente, é fácil perceber como cai o desempenho

do algoritmo como um todo quando a discretização aumenta, como já introduzido durante os exemplos. Por "algoritmo como um todo" entende-se configuração espacial mais busca (feita esta pelo **A***). A conseqüência desta queda de desempenho é a formação de um indesejável compromisso: eficácia (definido neste trabalho como qualidade da solução) *versus* eficiência (definida neste trabalho como o inverso do tempo total de busca): a consecução de um exclui, proporcionalmente, a obtenção do outro. Isto é indesejável, pois se deseja um algoritmo eficiente e eficaz, assim sendo estes dois não devem se excluir mutuamente.

O **SA** é mais custoso que o outro método em termos de processamento – e, portanto, de tempo total de busca. Entretanto, tal processamento requerido não cresce exageradamente com a complexidade do problema, do ambiente, como ocorre com alguns algoritmos não-exaustivos. Como cita amplamente a literatura, ele é ideal para a solução de problemas combinatórios complexos, problemas *NP-hard*, NP-complexos, de solução inviável via busca em grafos.

O Recozimento Simulado – *Simulated annealing* – pode, assim, ser uma ferramenta útil no planejamento da locomoção de robôs situados em espaços em que o cálculo de custo de trajetórias elementares é facilmente computável, mas que têm sua decomposição - do espaço - difícil. Esta é uma situação que surge em dois casos (os dois podem aparecer juntos ou não): primeiro, quando se lida com espaços com contornos definidos por curvas. Por "contornos" entende-se os obstáculos dentro do ambiente e também os limites do próprio espaço de trabalho. O segundo caso é quando o espaço tem ao menos um trecho com variação contínua de custo de travessia (i.e., custos de travessia definidos por funções do tipo *custo=custo(x,y)* ao invés de degraus de custo de um lugar do ambiente para outro).

LISTA DE REFERÊNCIAS

Aarts, E. H. L., van Laarhoven, P. J. M. (1985). Statistical Cooling: a General Approach to Combinatorial Optimization Problems. Philips J. Res., 40:193-226.

Abramson, D., Krishnamoorthy, M., Dang, H. (1997). Simulated Annealing Cooling Schedules for the School Timetabling Problem. Asia-Pacific Journal of Operational Research, Australia, 16:1-22.

Bonomi, E., Lutton, J.L. (1984). The n-city travelling salesman problem: Statistical mechanics and the Metropolis Algorithm. SIAM Review, 26(4):551-569.

Cohn, H., Fielding, M. (1999). Simulated Annealing: Searching for an Optimal Temperature Schedule. SIAM Journal of Optimization, Australia, 9(3):779-802.

Eckel, B. (1991). C++: Guia do Usuário. Makron, McGraw-Hill, Inc., São Paulo.

Elmohamed, M. A., Fox, G., Coddington, P. (1998). A Comparison of Annealing Techniques for Academic Course Scheduling. DHPC-045 and NPAC technical report SCCS-777, USA.

Evers, A. (1998). A Brief Introduction to Simulated Annealing. Author's web site, Australia.

Hajek, B. (1988) Cooling Schedules for Optimal Annealing. J. Math. Oper. Res., 13:311-329

Huang, M., Romeo, F., Sangiovanni-Vincentelli, A. (1986). An Efficient General Cooling Schedule for Simulated Annealing. Proc. of the IEEE International Conf. on Computer Aided Design (ICCAD):381-384.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. (1983). Optimization by Simulated Annealing. Science (220) pp.671-680.

Lafore, R. (1995). Object-oriented Programming in C++. 2nd Edition. Waite Group Press.

Latombe, J.C. (1993). Robot Motion Planning. New York: Kluwer Academic.

Lundy, M., Mees, A. (1986). Convergence of an Annealing Algorithm. Mathematical Programming, 34:111-124.

Lutfiyya, H. et al. (1992). Composite Stock Cutting Through Simulated Annealing. Mathematical and Computer Modelling. 16(1), 57-74.

Moraes, V.R. (1999). Simulador de Algoritmos de Otimização de Caminhos para AGVs. EPUSP.

Moraes, V.R., Martins T.C., Tsuzuki, M.S.G. (2001). Simulated Annealing Applied to Path Planning. 2nd IFAC Workshop on Intelligent Assembly and Disassembly (IAD2001).

Muñiz, F. et al. (1995). Neural Controller For A Mobile Robot In A Non-stationary Environment. University of Valladolid.

Nilsson, N.J. (1971). Problem-solving Methods in Artificial Intelligence. McGraw-Hill, Inc.

Nilsson, N.J. (1980). Principles of Artificial Intelligence. McGraw-Hill, Inc.

Ong, C.J., Gilbert, F. G. (1998). Robot Path Planning with penetrating growth distance. Journal of Robotic Systems. 15(2), 57-74.

Preiss, B. (1997). Data Structures and Algorithms with Object-oriented Design Patterns in C++. http://www.brpreiss.com/books/opus4/html/page479.html, EUA.

Schreiber, T., Schmitz, A., Richter, M. (1998). General constrained randomization. http://ls11-www.cs.uni-dortmund.de/people/hermes/NLDdocs/docs/wuppertal/docs/randomize_cool.html, Alemanha.

WIRTH, N. (1986). Algorithms & Data Structures. Prentice Hall, Inc., New Jersey.

Yang, X., Meng, M. (2000). An Efficient neural network approach to dynamic robot motion planning. Neural Networks. 13(2), 143-148.