Algoritmo Genético na otimização da adição de adjacências em um grafo com aplicação em planejamento do movimento de robôs

FELIPE A. W. BELO, MARCO A. MEGGIOLARO, MARCO A. PACHECO

ICA : Laboratório de Inteligência Computacional Aplicada

Departamento de Engenharia Elétrica

Pontifícia Universidade Católica do Rio de Janeiro, PUC-Rio

Rua Marquês de S. Vicente 225, Gávea, Rio de Janeiro, CEP 22453-900, RJ, Brasil

Palavras Chave: Inteligência Artificial, Algoritmos Genéticos, Robótica, Grafos, Planejamento do movimento.

1- Resumo

É apresentado o uso de um Algoritmo Genético para a solução do problema de adicionar novas adjacências em um grafo, buscando minimizar a soma das distâncias dos menores caminhos, para todos os nós. A aplicação deste problema aqui discutida é em um algoritmo para exploração robótica e modelagem de um ambiente. O problema também faz uso do algoritmo *A**, algoritmo clássico de busca em grafos.

O uso de Algoritmo Genético foi aplicado para dois grafos, com adição de diferentes números de adjacências. Também foi testado o uso da técnica para grafos onde inicialmente não existiam adjacências. Foram obtidos bons resultados e são propostas modificações a técnica aplicada de maneira a encontrar resultados ainda melhores.

2 - Introdução:

A modelagem em grafos pode ser empregada em diversos problemas de representação, tal como a descrição de redes, árvores diversas, e principalmente em esquemas de localização como mapas topológicos ou descrições geométricas de um ambiente.

Vários métodos de planejamento do movimento transformam um problema contínuo (ex. encontrar o melhor caminho entre dois pontos no espaço) em um problema discreto representado por um grafo, como pode ser visto em (Latombe, J.C. [1]). Muitos destes métodos envolvem busca em grafos.

Soluções para problemas de procura de um caminho em grafos são exploradas por vários métodos, como o *depth-first search* (Latombe, J.C. [1]) ou o algoritmo *A** (Hart, P.E. [6]), que será utilizado no projeto aqui apresentado.

Problemas de procura de um caminho em grafos também costumam ser abordados pela técnica de Algoritmos Genéticos. Um problema clássico da aplicação de Algoritmos Genéticos em grafos é o do "caixeiro viajante" (Pacheco M.A. [5]).

Outras aplicações da técnica de Algoritmos Genéticos em problemas envolvendo grafos podem ser encontradas, muitas delas com bons resultados. Exemplos podem ser encontrados em (Pacheco M.A [5]) ou (Mitchell M. [3]).

O problema aqui abordado é o da adição de novas adjacências em um grafo, buscando minimizar a soma dos menores caminhos, para todos os nós. Pretende-se usar a solução aqui implementada junto do algoritmo para exploração robótica e modelagem de um ambiente proposto em (Sujan V.A. e Meggiolaro M.A. [2]).

O problema pode ser visto do seguinte modo, dado um grafo existente, pretende-se adicionar *n* novas adjacências ao grafo, buscando otimizar a soma das distâncias percorridas pelos menores caminhos entre cada par de nós do grafo.

Aplicações que utilizam representações em grafos poderiam se beneficiar da solução deste problema. Alguns exemplos são o projeto de rodovias (construção de novas rodovias), rotas aéreas ou redes de computadores.

A aplicação que este projeto visa, é do uso do método aqui abordado, na escolha de caminhos a serem percorridos por um robô que trabalha com a modelagem do ambiente através de grafos, onde cada nó do grafo representa uma posição no espaço 2-D *(x,y)*, e cada adjacência representa um caminho percorrido pelo robô. Esta aplicação será aprofundada em *3.1*.

Para solucionar o problema proposto foram utilizadas técnicas de Algoritmo Genético em conjunto com técnicas de planejamento do movimento e busca em grafos.

Foi desenvolvido um software para geração dos grafos usados nos testes, uso do algoritmo *A** e para rodar o Algoritmo genético com os parâmetros desejados.

3 – Adicionando adjacências em um grafo

3.1 – A Aplicação: Modelagem e Exploração Robótica de um Ambiente Baseado em Informação Usando Imagens 2-D e Grafos

Pretende-se utilizar a solução do problema aqui proposto junto do algoritmo de exploração robótica e modelagem de um ambiente apresentado em (Sujan V.A. e Meggiolaro M.A. [2]), este, aqui brevemente descrito. Porém, esta não é a única aplicação que poderia se beneficiar deste problema. Alguns outros exemplos de aplicações possíveis são o projeto de rodovias (construção de novas rodovias), rotas ou redes.

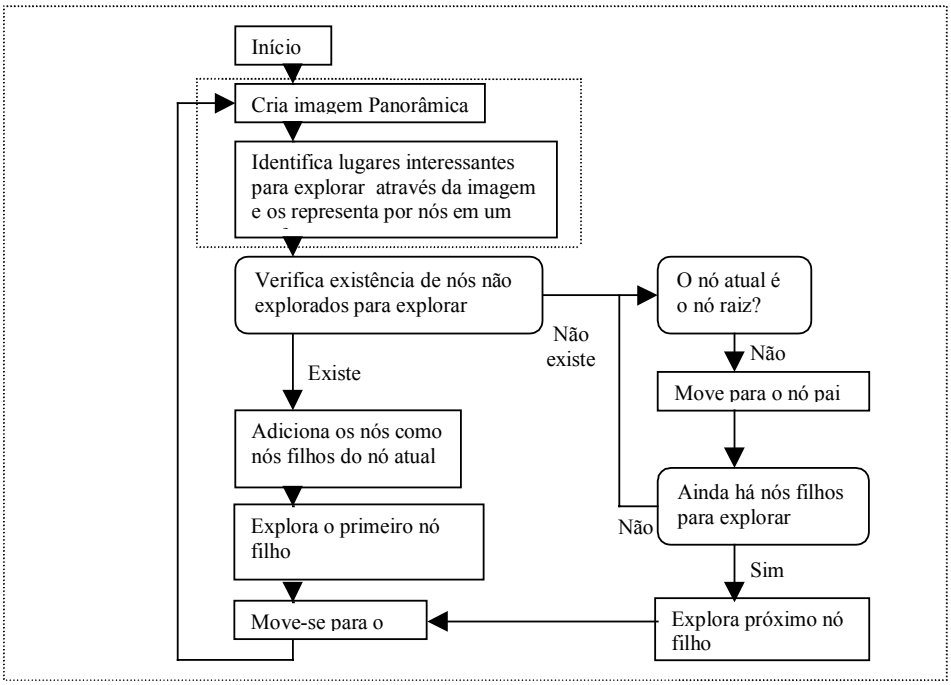
A interação mais básica de um robô com o ambiente é feita pela percepção deste e navegação através do mesmo. Para navegar, o robô precisa de um modelo do ambiente, sem o qual, não pode tomar decisões. Em determinadas aplicações, o modelo não é dado ao robô, este precisa então, fazer a modelagem do ambiente movendo-se e "sentindo" o mesmo. A tarefa de navegação pode ser então compreendida de um modo genérico através de duas etapas: modelagem do ambiente, e navegação.

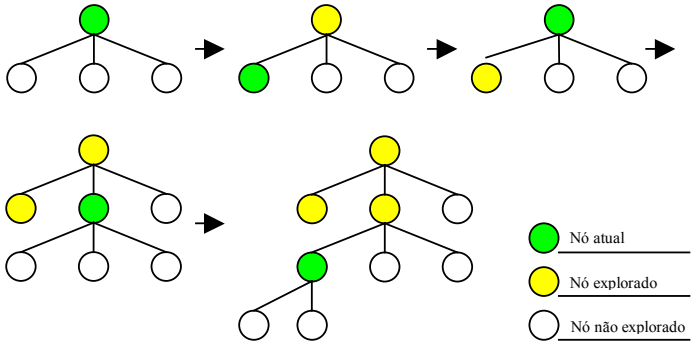
As propostas de modelagem e navegação em um ambiente aqui descritas são uma alternativa para a redução do custo de robôs móveis autônomos para uso indoor. Estes robôs utilizariam de uma câmera e de sensores de toque para fazer a modelagem do ambiente, trabalhando-se com um algoritmo para planejamento da estratégia de exploração visual.

O algoritmo de modelagem proposto se diferencia da grande maioria dos algoritmos encontrados por não fazer a modelagem através de um mapeamento usual, e sim, através da definição e exploração de posições

no ambiente, escolhidas através da análise visual do mesmo.

O algoritmo de modelagem do ambiente pode ser entendido abaixo:





O algoritmo de exploração está exemplificado no esquema acima.

Por não agregar muito ao projeto aqui desenvolvido, não será abordado como é feita a escolha de novos nós a serem explorados. Para maiores detalhes, verificar em em (Sujan V.A. e Meggiolaro M.A. [2]).

O modelo criado pelo robô é uma árvore de nós. Cada adjacência carrega a informação de distância entre dois nós e o ângulo entre eles. Cada nó guarda a imagem panorâmica criada naquele ponto do espaço. Esta modelagem possibilitará uma futura navegação baseada em busca de imagens, seja de objetos, lugares, ou outros.

Em uma segunda etapa da modelagem, podemos melhorar o modelo criado. Para tal, buscam-se novas adjacências entre os nós encontrados. Desta maneira, quando houver navegação, o robô não necessitará percorrer muitos caminhos desnecessários para chegar a algum lugar escolhido.

Para criar novas adjacências, o robô precisará tentar percorrer o caminho entre os dois nós que a compõem. Em alguns casos, este caminho pode conter obstáculos. Como esta tarefa requer tempo, seria impensado tentar criar todas as adjacências

possíveis, para todos os nós. Portanto, seria interessante escolher as adjacências a serem percorridas. Esta escolha seria feita de maneira a otimizar a futura navegação. Aqui entra o problema de adicionar novas adjacências em um grafo.

3.2 – Descrição do problema de adicionar novas adjacências em um grafo

Dado o grafo *G = (X, A)*, composto do conjunto de nós *X* e o conjunto de adjacências *A*, deseja-se definir um conjunto *A'* de *n'* novas adjacências para acrescentá-lo a *G*, criando-se um novo grafo *G'(X,A+A')*. A escolha do conjunto *A'* deve ser feita de maneira a minimizar a soma das distâncias dos menores caminhos entre todos os nós:

$$F(x_1, x_2, ..., x_n) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} g(x_i, x_j)}{2}$$

Onde x_i representa o nó *i* de *X*, e $g(x_i,x_j)$ é uma função que retorna a distância do menor caminho entre os nós x_i e x_j.

A função *g(x,y)* é calculada utilizando o algoritmo *A** que encontra o menor caminho entre dois nós de um grafo.

O conjunto de nós *X* foi definido neste projeto como pontos em um plano cartesiano com coordenadas *(x,y)*, pois os grafos trabalhados representam um modelo de um ambiente em 2-D. A distância associada a uma adjacência é dada pela distância entre os dois nós que esta adjacência liga:

$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \qquad (3.2)$$

*3.3 – O Algoritmo A**

Para um grafo *G* qualquer, quando as adjacências em *G* possuem um custo e o custo de um caminho é definido como a soma dos custos relativos às adjacências que compõem o caminho, pode ser interessante definir o caminho de menor custo entre um nó inicial N_{init} e um nó objetivo N_{goal}. Um algoritmo clássico para gerar este caminho é o *A**.

*O A** explora *G* iterativamente, percorrendo caminhos originados em N_{init}. No início de cada iteração, há nós que o algoritmo já visitou e outros ainda não visitados. Para cada nó *N* visitado, as iterações anteriores produziram um ou mais caminhos conectando N_{init} a *N*, porém o algoritmo só guarda a representação do caminho de menor custo. A qualquer momento, o conjunto de tais caminhos forma uma árvore *T* com o grupo de nós explorados. T é representada associando a cada nó visitado *N* (exceto N_{init}) um ponteiro ao nó parente.

*O A** associa uma função custo *f(N)* para cada nó *N* em *T* atual. Esta função é uma estimativa de custo do custo mínimo para o caminho entre N_{init} e N_{goal} através de *N*:

$$f(N) = g(N) + h(N)$$

Onde:

- *g(N)* é o custo do caminho entre N_{init} em *N* em *T* atual;
- *h(N)* é uma estimativa heurística do custo *h*(N)* do custo do menor caminho entre *N* e N_{goal};

A função *h* é dita admissível se e somente se satisfizer: *(3.1)*

$$\forall N \in G : 0 \le h(N) \le h^*(N)$$

Para o problema em questão poderia se fazer *h(N)* igual à distância cartesiana entre *N* e N_{goal} , o que provavelmente faria o algoritmo rodar mais rápido, porém foi adotado *h(N) = 0*. Fica proposto o uso de *h(N)* igual à distância cartesiana.

O algoritmo *A** é dado no apêndice A.

4 – Aplicando o Algoritmo Genético

Para encontrar *n* novas adjacências a serem adicionadas em um grafo, será empregada um Algoritmo Genético com o objetivo de minimizar a soma das distâncias mínimas entre todos os nós após a adição das novas adjacências.

4-1 – Representação do cromossomo

O cromossomo utilizado deve representar as novas adjacências a serem criadas. Algumas possíveis representações são:

- Máscara de bits: Cada gene do cromossomo está associado à uma adjacência do grafo e representa sua adição ao grafo. Se o bit estiver ligado, aquela adjacência é adicionada, se desligado, não. O tamanho do cromossomo deve ser de $k^2/2$, onde k é o número de nós do grafo. O espaço de busca é dado por $2^{k^2/2}$. No exemplo seguinte pode-se ver um cromossomo onde há 4 possíveis adjacências, e a primeira e a última serão adicionadas ao grafo. Ex: (1-0-0-1);
- Representação por inteiros: Os genes são números inteiros representando os nós do grafo. Cada

adjacência a ser adicionada ao grafo é representada por um par de genes. O tamanho do cromossomo deve ser de 2n, onde n é o número de nós a serem adicionados ao cromossomo. O espaço de busca é dado por k^{2n}. No exemplo seguinte, n = 2 e as adjacências entre os nós 1 e 2 e entre os nós 5 e 8 são adicionadas. Ex: ((1 - 2) , (5 - 8));

A segunda representação foi a escolhida para este projeto. Perceba que diferentes cromossomos podem representar a mesma solução, atrapalhando o desempenho do algoritmo.

4-2 Operadores utilizados

Somente dois operadores foram utilizados, crossover e mutação.

O operador de crossover utilizado foi o crossover de um ponto. Dado um ponto aleatório entre dois genes, os cromossomos pais são divididos e recombinados para gerar os cromossomos filhos. Outros operadores de crossover possíveis são o crossover de dois pontos e o crossover uniforme.

O operador de mutação utilizado, sorteava a troca do valor de um gene por um valor aleatório. É proposto, apesar de não utilizado, fazer uma mutação por adjacência. Esta mutação se daria da seguinte maneira. Sorteado um gene para ocorrer mutação, seu novo valor seria um nó sorteado dentre os nós vizinhos ao nó representado por tal gene.

4-3 Método de seleção

O método de seleção utilizado foi o método de seleção por roleta, com uso de normalização e *Steady-State* com duplicados.

4-4 Avaliação do A.G.

Para avaliar os cromossomos gerados, são adicionadas as adjacências ao grafo *G* trabalhado. e então, este grafo é avaliado segundo a seguinte métrica denominada *Eficiência do Grafo*:

$$f(G) = \frac{D(x_1, x_2, ..., x_n)}{F(x_1, x_2, ..., x_n)}$$

Sendo a função *D*, somatório das distâncias cartesianas entre todos os nós. dada por:

$$D(x_1, x_2, ..., x_n) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} d(x_i, x_j)}{2}$$

A função *f* será máxima e igual a 1 quando ou todos os nós estiverem ligados, ou o caminho mínimo para todos os nós seja uma reta.

Caso algum dos nós não tenha adjacência com nenhum dos outros nós, a função de avaliação retorna 0. Isto permite que o algoritmo genético seja utilizado para gerar adjacências em grafos sem adjacências. Desta maneira o A.G. pode solucionar problemas como o de encontrar o melhor conjunto de rotas para um conjunto de localizações. Porém, apesar de possibilitar tal uso, esta avaliação não é a melhor para este tipo de problema, pois o algoritmo genético pode demorar muito tempo para encontrar soluções validas (que não retornem $f = 0$) sem conseguir encontrar bons schematas, dado que todos os cromossomos que representam soluções inválidas retornam o mesmo valor, sem diferenciação.

5 – O Software desenvolvido

Para testar a solução proposta foi desenvolvido um software em C++ utilizando o programa *Microsoft Visual C++ 6.0*. Este software permite gerar os grafos testados e visualizá-los, verificar as métricas utilizadas, rodar o algoritmo *A** e rodar o Algoritmo Genético. Além disto, o software gera curvas de desempenho do Algoritmo Genético e possibilita salvar os resultados para serem visualizados em outros programas, como o *Microsoft Excel* por exemplo.

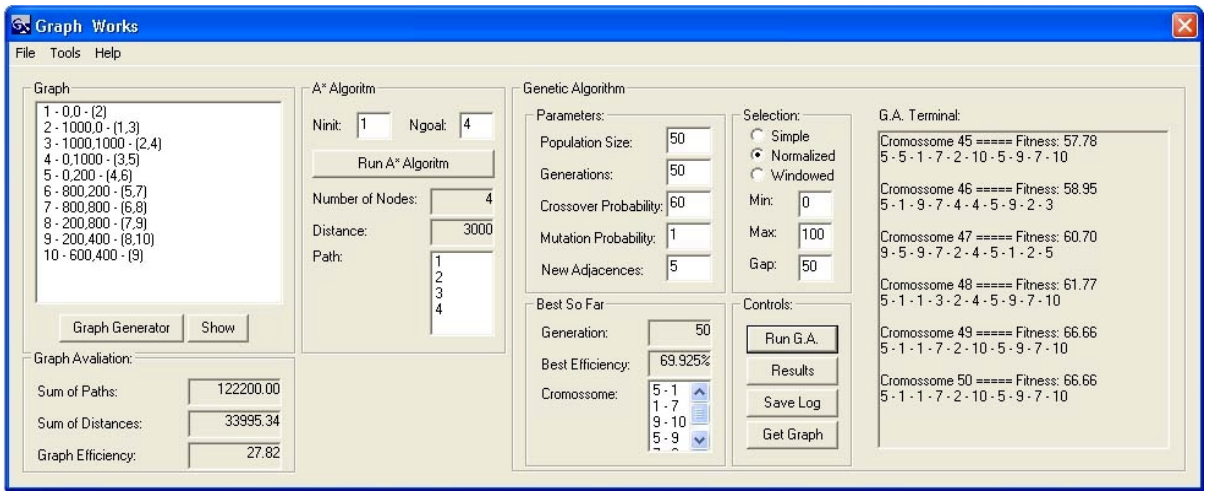


Fig 5.1 Interface do software desenvolvido

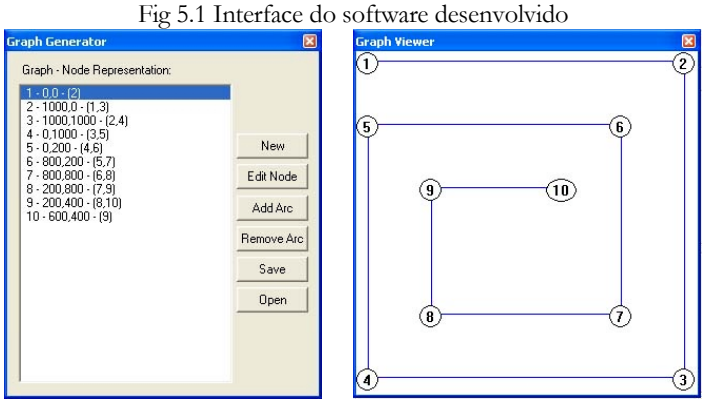


Fig 5.2 Interface das tools *Generator* e *Viewer*

Este software foi feito com as seguintes limitações:

- Grafos com o máximo de 100 nós;
- Espaço cartesiano limitado em $0 \le x \le 1000, 0 \le y \le 1000$;

6 – Experimentos e discussão dos resultados

Para testar o desempenho do Algoritmo Genético, este foi empregado na adição de adjacências para dois grafos diferentes, um de 10 nós, outro de 25 nós, denominados respectivamente de grafo *Espiral* e grafo *Robô* (Fig. 6.1 *a* e Fig 6.3 *a*). Para alguns dos testes, o desempenho do A.G. é comparado ao desempenho de uma busca aleatória. Foram realizados alguns testes para encontrar-se a melhor combinação de parâmetros para o A.G. e são apresentados aqui testes realizados com a seguinte configuração: População de 100 indivíduos / Gap de 90 indivíduos / Crossover em 80% / Mutação em 5% / Normalização entre 0 e 100.

Também foram feitos 2 testes com os mesmos grafos porém sem adjacências (Figuras 6.5 *a* e *b*), procurando-se adicionar o número mínimo de adjacências de maneira a cobrir todos os nós e encontrar o melhor resultado.

Devido ao fato do processamento do A.G. ser lento, principalmente para um número de nós elevado, os resultados apresentados são relativos a somente um teste para cada experimento, e não a uma média relativa a vários experimentos. Os testes apresentados foram escolhidos dentre vários testes realizados por sua maior relevância quanto à discussão dos resultados obtidos.

6.1 – Grafo "Espiral"

O primeiro grafo a ser apresentado possui 10 nós e tem o formato de uma espiral. As figuras 5.1 e 5.2 apresentam respectivamente os melhores resultados obtidos para algumas adições de adjacências e gráficos comparativos da evolução do A.G. e busca aleatória. Perceba que o número de gerações varia de acordo com as experiências devido ao fato de algumas levarem mais tempo para convergirem.

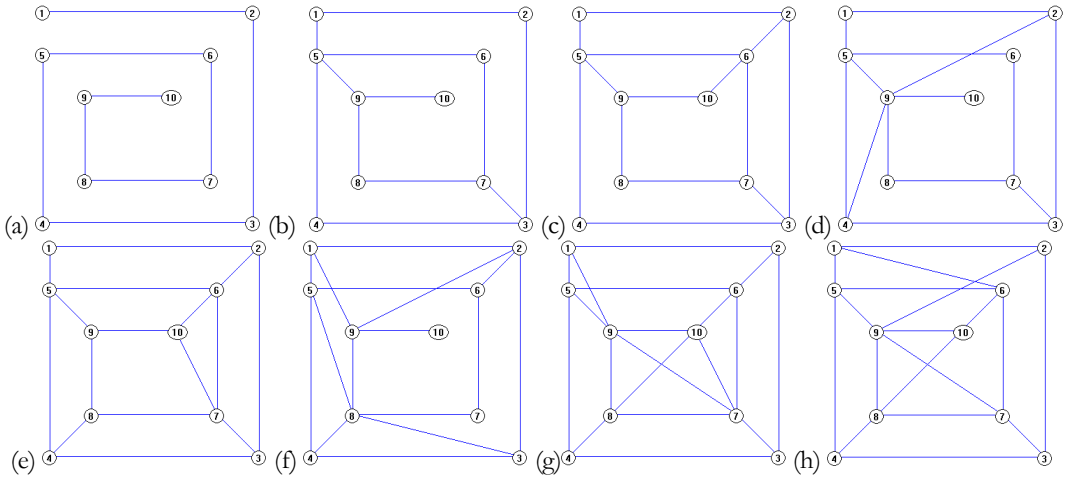


Fig. 6-1: (a) Grafo *Espiral* sem adição de adjacências, (b) Adição de 3 adjacências pelo A.G e busca aleatória, (c) Adição de 5 adjacências pelo A.G, (d) Adição de 5 adjacências pela busca aleatória, (e) Adição de 7 adjacências pelo A.G, (f) Adição de 7 adjacências pela busca aleatória, (g) Adição de 10 adjacências pelo A.G, h) Adição de 10 adjacências pela busca aleatória.

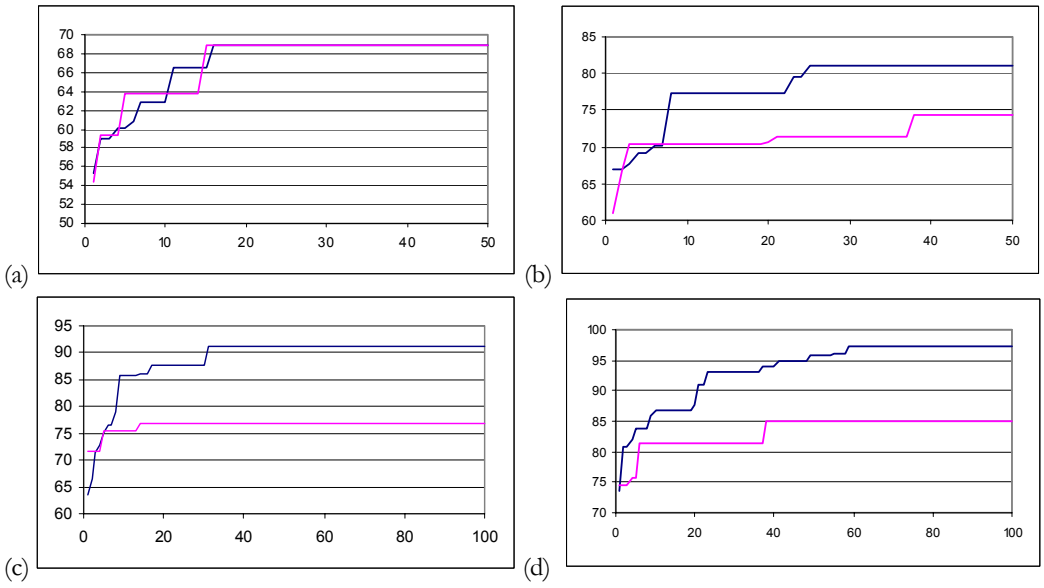


Fig. 6.2: Aptidão dos melhores indivíduos x Gerações para grafo *Espiral* - (a) Adição de 3 adjacências, (b) Adição de 5 adjacências, (c) Adição de 7 adjacências, (d) Adição de 10 adjacências. OBS: As curvas rosas são referentes à busca aleatória e as curvas azuis são referentes ao A.G.

Pode-se verificar que o A.G. apresenta excelentes resultados para grafos com pequeno número de nós em poucas gerações. A superioridade do A.G. em relação à busca aleatória se mostra mais claramente com um número maior de adjacências acrescentadas ao grafo. Isto se deve ao fato de que quando o espaço de busca é maior, a busca aleatória é lenta e pouco eficaz. Para a adição de poucas adjacências, não é verificada grande diferença entre o A.G. e a busca aleatória.

Para os testes efetuados com adições de até 7 adjacências, o A.G. demonstrou encontrar soluções ótimas. Para a adição de 10 adjacências, o resultado encontrado é sub-ótimo. Isto acontece, entre outros possíveis motivos, pois foi percebido ao acompanhar os cromossomos criados, que os melhores indivíduos tendiam a se repetir ao longo das gerações, levando a

uma convergência precipitada do algoritmo para configurações sub-ótimas. Para resolver tal problema ficam propostas algumas técnicas:

- Trabalhar com representações do problema onde diferentes cromossomos não representam o mesmo resultado;
- Trabalhar com Steady State sem duplicados. na implementação feita, o uso de Steady State leva a um grande número de indivíduos replicados, fazendo com que não exista grande variação nas populações criadas, o que leva à convergência do algoritmo;
- Interpolar os parâmetros do A.G. A interpolação dos parâmetros permite que quando o algoritmo estiver convergindo para um resultado sub-ótimo, se diversifique a população, seja aumentando a probabilidade de mutação do algoritmo, seja diminuindo a pressão seletiva sobre os melhores indivíduos;
- Incluir novos operadores ao A.G. que permitam seu melhor desempenho e possibilitem uma diversificação da população de forma mais eficaz. Um possível operador é o de mutação por adjacências já apresentado em *3.2 Operadores Utilizados;*.
- Rodar o algoritmo algumas vezes, semeando as novas populações com os melhores indivíduos dos experimentos anteriores;

6.2 – Grafo "Robô"

O segundo grafo apresentado possui 25 nós e tem o formato de uma árvore. Este grafo é uma possibilidade de modelo retornado por um robô após a exploração do ambiente e por isso foi denominado *Robô*.

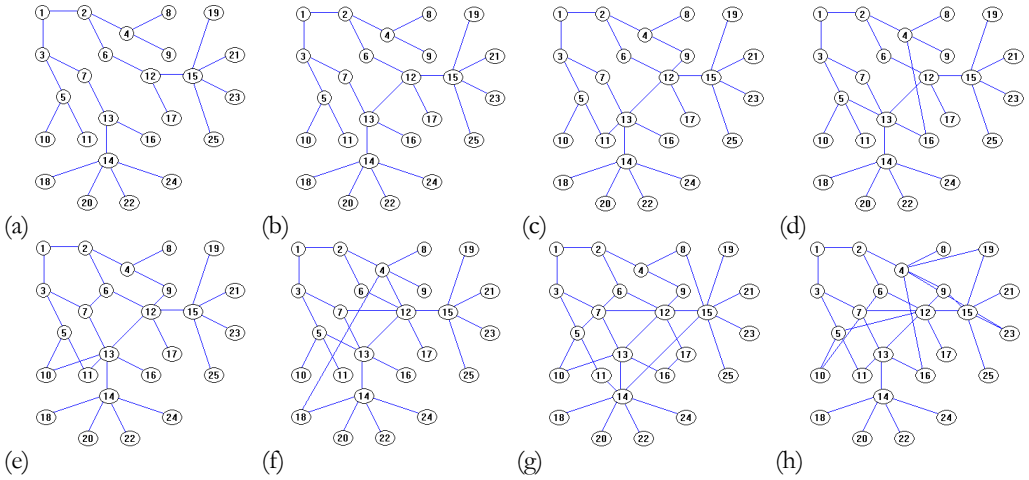
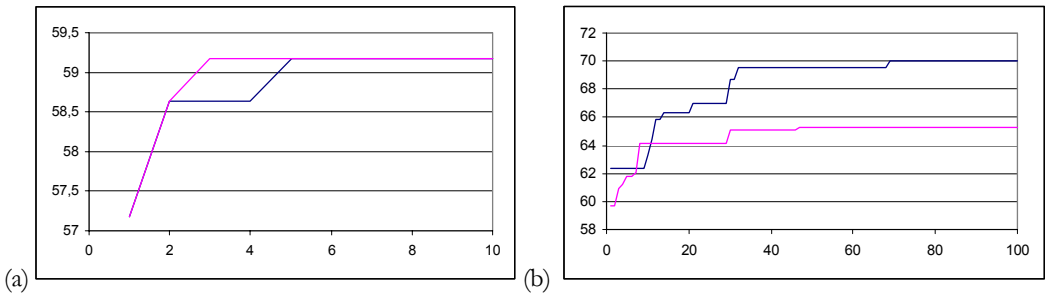


Fig. 6.3: (a) Grafo *Robô* sem adição de adjacências, (b) Adição de 1 adjacência pelo A.G e busca aleatória, (c) Adição de 3 adjacências pelo A.G, (d) Adição de 3 adjacências pela busca aleatória, (e) Adição de 5 adjacências pelo A.G, (f) Adição de 5 adjacências pela busca aleatória, (g) Adição de 10 adjacências pelo A.G, h) Adição de 10 adjacências pela busca aleatória.



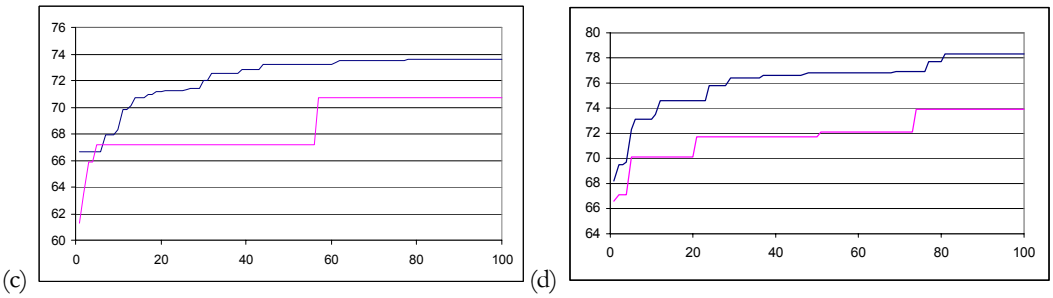


Fig. 6.4: Aptidão dos melhores indivíduos x Gerações para grafo *Robô* - (a) Adição de 1 adjacência, (b) Adição de 3 adjacências, (c) Adição de 5 adjacências, (d) Adição de 10 adjacências. OBS: As curvas rosas são referentes à busca aleatória e as curvas azuis são referentes ao A.G.

Novamente pode-se perceber que o Algoritmo Genético retornou bons resultados. Como para o grafo *Espiral*, com o aumento do número de adjacências acrescentadas, e conseqüentemente, o aumento do cromossomo, os resultados obtidos não são mais ótimos. Porém, para conseguir melhores resultados, podem ser aplicadas as idéias propostas anteriormente.

Os comentários relativos ao grafo *Espiral* se aplicam ao grafo R*obô*.

6.3 – Grafos sem arcos

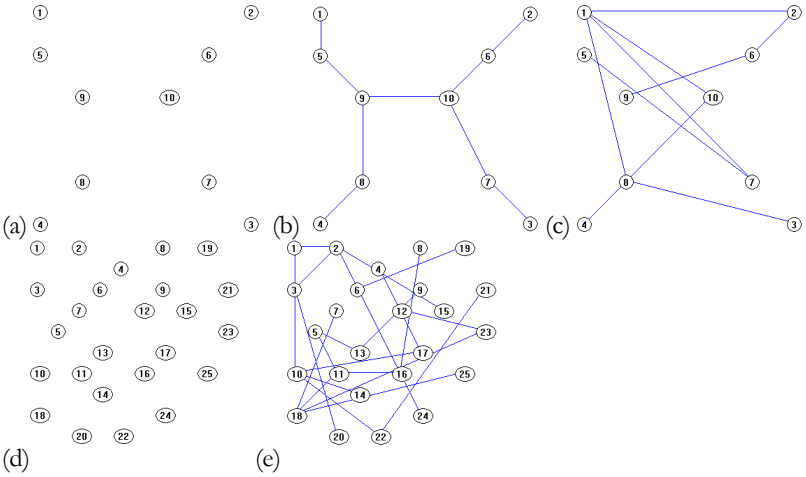
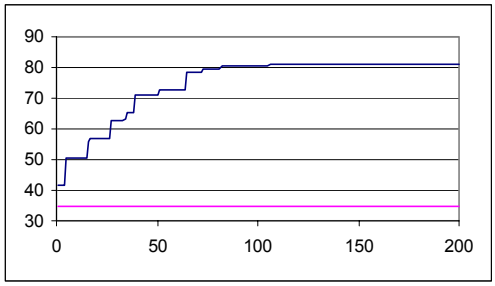


Fig. 6.5: (a) Grafo *Espiral Sem Arcos* sem adição de adjacências, (b) Grafo *Espiral Sem Arcos* com adição de 1 adjacência pelo A.G, (c) Grafo *Espiral Sem Arcos* com adição de 1 adjacência pela busca aleatória, (d) Grafo *Robô Sem Arcos* sem adição de adjacências, (e) Grafo *Robô Sem Arcos* com adição de 1 adjacência pelo A.G.

(a) 

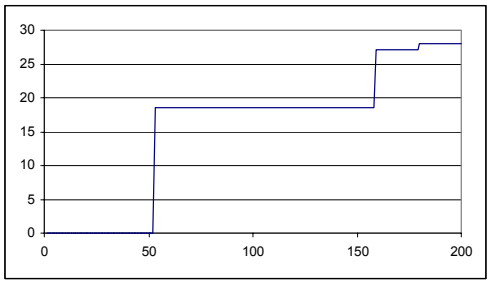
(b) 

Fig. 6.6: Aptidão dos melhores indivíduos x Gerações - (a) Adição de 9 adjacências no grafo *Espiral sem Arcos*, (b) Adição de 24 adjacências no grafo *Robô sem arcos*. OBS: As curvas rosas são referentes à busca aleatória e as curvas azuis são referentes ao A.G.

Ao se utilizar o Algoritmo Genético como proposto, buscando resolver o problema de escolher a melhor combinação para um número mínimo de adjacências, dado um conjunto de nós, foi encontrada boa solução para o caso de 10 nós. Porém, o algoritmo não obteve boa evolução para o conjunto de 25 nós testado. O mau resultado se deve ao fato da função de avaliação retornar 0 para cromossomos que não representam soluções possíveis, fazendo com que o algoritmo demore em encontrar cromossomos viáveis e não possa desenvolver melhores cromossomos até então. Para resolver este problema, fica proposto gerar uma população inicial de indivíduos que já contenham cromossomos que representem soluções plausíveis, ou, seja alterada a função de avaliação de modo a conseguir avaliar estes cromossomos.

7 – Conclusão

O problema de adicionar novas adjacências a um grafo é tratado através de um Algoritmo Genético gerando bons resultados. Percebe-se que quanto maior o número de nós, maior a dificuldade associada ao problema, e melhores os resultados obtidos pelo Algoritmo Genético em relação à busca aleatória. A função de avaliação utilizada permitiu ainda o uso do A.G. para tentar resolver o problema de escolher a melhor combinação para um número mínimo de adjacências. Porém, os resultados obtidos só foram bons para um número pequeno de nós. Para resolver o problema de um grande número de nós, seria necessário fazer modificações na função de avaliação ou semear a população inicial com indivíduos válidos.

São propostas diversas sugestões visando resultados ainda melhores, entre elas:

- Uso de Steady State sem duplicados;
- Trabalhar com diversos experimentos, semeando novas populações com os melhores indivíduos de populações anteriores;
- Interpolar os parâmetros do A.G.;
- Trabalhar com operadores diferentes dos utilizados;

Verificou-se a possibilidade da aplicação do método implementado não somente na aplicação proposta em robótica mas também no planejamento de rodovias, rotas aéreas ou redes de computadores, como alguns exemplos.

Bibliografia:

[1] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
[2] Vivek A. Sujan, Marco A. Meggiolaro. *Information Based Indoor Environment Robotic Exploration and Modeling Using 2-D Images and Graphs.*
[3] Melanie Mitchell. *An Introduction to Genetic Algorithms.* Mit Press, 1999.
[4] Marco Aurélio Cavalcanti Pacheco. *Algoritmos Genéticos: Princípios e Avaliações*.
[5] Marco Aurélio Pacheco, ***Notas de Aula em Computação Evolucionária***, (www.ica.ele.puc-rio.br).
[6] Hart, P.E., Nilsson, N.J e Raphael, B. *A Formal Basis for the Heuristic Determination of Minimal Cost Paths,* IEEE Transactions on Systems, Science and Cybernetics SSC-4(2), 100-107, 1968.

Apêndice A – Algoritmo A*

As entradas consistem em *G*, N_{init} , N_{goal} , *k*, e *h*, onde $k : X \times X \rightarrow R^{+}$ é a função que especifica a distância de cada adjacência em *G*. Todos os nós são inicialmente marcados como não visitados. O algoritmo faz uso de uma lista denominada *OPEN* que suporta as seguintes operações:

- *FIRST(OPEN)*: Remove o nó de *OPEN* com o menor valor de *f* e retorna o mesmo;
- *INSERT(N,OPEN)*: Insere o nó *N* em *OPEN*;
- *DELETE(N,OPEN)*: Remove o nó *N* de *OPEN*;
- *MEMBER(N,OPEN)*: Retorna *verdadeiro* se *N* estiver em *OPEN* e *falso* caso contrário;
-*EMPTY(OPEN)*: Retorna *verdadeiro* se *OPEN* estiver vazio e *falso* caso contrário;

```
procedimento A*(G, Ninit , Ngoal , k, h)
    inicio
        insira Ninit em T;
        INSERT(N_init , OPEN);
        marcar N_init como visitado;
        while ! EMPTY(OPEN) do
            N = FIRST(OPEN)
            if N = N_goal then sair do loop while;
            for todo nó N' adjacente a N em G do
                if N' não visitado then
                    adicionar N' em T com ponteiro para N;
                    f(N) = f(N) + k(N,N');
                    INSERT(N',OPEN);
                    marcar N' como visitado;
                    fim;
                else if f(N') > g(N) + k(N,N') then
                    f(N) = f(N) + k(N,N');
                    modificar T redirecionando o ponteiro de N' para N
                    fim;
                fim
            fim
        if N = N_goal then
            retorna o caminho construido traçando os ponteiros em T de N_goal a N_init;
        else retorna falha;
    fim
```