GARDÊNIO PUIATTI RODRIGUES

OTIMIZAÇÃO DE ROTAS ATRAVÉS DA APLICAÇÃO DE ALGORITMOS EXATOS E HEURÍSTICOS

Trabalho de conclusão de curso apresentado ao Curso de Ciência da Computação.

UNIVERSIDADE PRESIDENTE ANTÔNIO CARLOS

Orientador: Prof. Eduardo Bhering

BARBACENA
2004

GARDÊNIO PUIATTI RODRIGUES

OTIMIZAÇÃO DE ROTAS ATRAVÉS DA APLICAÇÃO DE ALGORITMOS EXATOS E HEURÍSTICOS

Este trabalho de conclusão de curso foi julgado adequado à obtenção do grau de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação da Universidade Presidente Antônio Carlos.

Barbacena – MG, 14 de junho de 2004.

_____

Prof. Eduardo Bhering - Orientador do Trabalho

_____

Prof. Gustavo Campos Menezes - Membro da Banca Examinadora

_____

Prof. Frederico de Miranda Coelho - Membro da Banca Examinadora

AGRADECIMENTOS

Agradeço a Deus, ao empenho de meu orientador, e ao apoio de familiares e amigos.

RESUMO

Este trabalho apresenta algumas técnicas para resolução de problemas relacionados à otimização utilizando algoritmos exatos como (Branch-and-Bound) e heurísticas como Algoritmos Genéticos, e Vizinho Mais Próximo, sendo o Algoritmo Genético o foco do estudo. O problema se resume em dado um conjunto de rotas obter a melhor rota, com o menor tempo possível de processamento, visando minimizar os custos e otimizar os resultados alcançados.

Palavras-chave: Otimização, Algoritmos Genéticos, Branch-and-Bound e Vizinho Mais Próximo.

SUMÁRIO

LISTAS

1 INTRODUÇÃO

Serviços do setor público, como por exemplo, varredura de ruas, coleta de lixo, roteamento de carteiros, inspeção de linhas de água, eletricidade ou gás, são realizados a partir de uma grande utilização de recursos humanos, visto que, para a execução dessas tarefas, é necessário haver a caminhada ao longo do trecho trabalhado.

Os pontos a serem visitados podem ser modelados através de um grafo, onde os pontos são vértices, e o trajeto de um vértice a outro como arestas destes vértices.

A otimização do trajeto a ser percorrido e a divisão em rotas dos trechos a serem atendidos, deve ser realizada respeitando as diversas restrições existentes, como velocidade de deslocamento, distância máxima percorrida, número mínimo e máximo de unidades atendidas, cobertura de todas as unidades, entre outras.

Uma maneira de se resolver este tipo de problema, é feita da seguinte maneira: A rota que se tem que percorrer para inspeção de linhas de água por um leiturista, é feita pelo funcionário mais experiente do setor, ou seja, a rota é traçada de acordo com a experiência de um funcionário.

Este tipo de problema se encaixa dentro de uma importante classe de problemas dentro da área de pesquisa de otimização combinatória, e pela sua importância tem recebido

especial atenção pelos pesquisadores da área. Existem várias soluções propostas na literatura para a resolução desse tipo de problema, como os algoritmos genéticos.

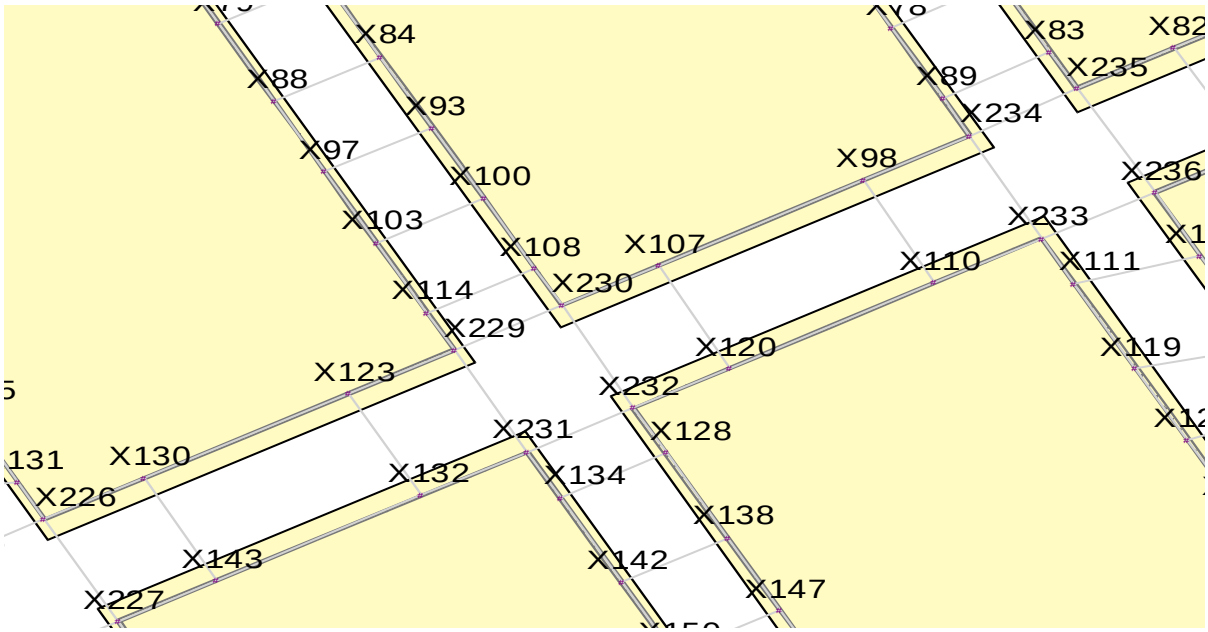
Os algoritmos genéticos surgem como uma metaheurística e vêm sendo aplicados nos mais diversos problemas de otimização, como localização de facilidades e roteamento de veículos . Esse tipo de algoritmo evolutivo destaca-se pela sua simplicidade de operação e implementação, aliada ao baixo tempo gasto na obtenção de uma solução para o problema.

1.1 OBJETIVO

Diante destes problemas de otimização, este trabalho se restringe a resolver um problema que envolve a procura de um melhor caminho, ou seja, dado um ponto inicial deve-se percorrer todos os pontos subseqüentes, sem que nenhum ponto deixe de ser visitado, e retorne a ponto de partida. O resultado da busca deve conter a melhor solução.

Conforme demonstrado na Figura 1, pode-se verificar qual o caminho a ser percorrido pelo leiturista, sendo que o mesmo, deverá fazer a escolha entre continuar no mesmo passeio ou atravessar a rua, ao final do percurso deverá retornar ao ponto inicial.

Figura 1 - Exemplo de um percurso:



2 OTIMIZAÇÃO

Otimizar é melhorar o que já existe, projetar o novo com mais eficiência e menor custo.[1]

A otimização visa determinar a melhor configuração de projeto sem ter que testar todas as possibilidades envolvidas.[1]

Problemas de otimização são caracterizados por situações em que se deseja maximizar ou minimizar uma função numérica de várias variáveis, num contexto em que podem existir restrições. Tanto as funções acima mencionadas como as restrições dependem dos valores assumidos pelas variáveis de projeto ao longo do procedimento de otimização. Pode-se aplicar otimização em várias áreas, como por exemplo, no projeto de sistemas ou componentes, planejamento e análise de operações, problemas de otimização de estruturas, otimização de forma, controle de sistemas dinâmicos.

Os problemas de otimização são baseados em três pontos principais: a modelagem do problema, a função objetivo que se deseja maximizar ou minimizar e o espaço de soluções associado. Pode-se imaginar um problema de otimização como uma caixa preta com *n* botões, onde cada botão é um parâmetro do problema, e uma saída que é o valor da função objetivo,

indicando se um determinado conjunto de parâmetros é bom ou não para resolver este problema.[1]

A otimização tem como vantagens diminuir o tempo dedicado ao projeto, possibilitar o tratamento simultâneo de uma grande quantidade de variáveis e restrições de difícil visualização gráfica e/ou tabular, possibilitar a obtenção de algo melhor, obtenção de soluções não tradicionais, menor custo.[1]

Como limitações, tem-se o aumento do tempo computacional quando se aumenta o número de variáveis de projeto, pode-se surgir funções descontínuas que apresentam lenta convergência, funções com presença de muitos mínimos locais onde o mínimo global raramente é obtido. [1]

Assim, os estudos de métodos heurísticos, com busca randômica controlada por critérios probabilísticos, reaparecem como uma forte tendência nos últimos anos, principalmente devido ao avanço dos recursos computacionais, pois um fator limitante destes métodos é a necessidade de um número elevado de avaliações da função objetivo (Schwefel e Taylor, 1994). Os algoritmos genéticos são mecanismos de busca baseada nos processos de seleção natural da luta pela vida e da genética de populações. Trata-se de um método pseudoaleatório, portanto pode-se dizer que é um método, um procedimento de exploração inteligente, no espaço de parâmetros codificados (*apud* BRAGA,98).[1]

O surgimento dos algoritmos genéticos deu-se por volta de 1950 quando vários biólogos usavam técnicas computacionais para a simulação de sistemas biológicos. Entre 1960 e 1970, na Universidade de Michigan, sob a direção de John Holland (1975), iniciou-se o estudo de algoritmos genéticos como os conhecidos atualmente. David Goldberg (1989) apresentou a solução de complexos problemas de engenharia usando algoritmos genéticos, o que ajudou o método a se tornar popular entre os pesquisadores. [1]

2.1 PROBLEMA DO CAIXEIRO VIAJANTE

Considere-se um grafo em que os vértices representam cidades e as arestas representam estradas de uma dada região (a cada estrada está associada uma distancia entre as

cidades). O problema é calcular o ciclo de menor distância total que, tendo inicio e fim na cidade “x”, passe em todas as cidades uma única vez.[2]

Este problema, é designado por problema do caixeiro viajante (*travelling salesman problem*), implica calcular no grafo um ciclo de Hamilton de encargo total mínimo. O tempo necessário para resolução deste problema cresce exponencialmente com o numero de vértices do grafo. Só o método de enumeração (identificação de todos os caminhos possíveis) garante o cálculo da solução ótima do problema, mas tal método é impraticável. Veja-se um computador for capaz de processar cerca de 10000 caminhos por segundo, necessitará aproximadamente de 18 segundos para calcular um percurso ótimo num grafo com 10 vértices, 50 dias para um grafo com 15 vértices, 2 anos para um grafo de 16 vértices e 19300 anos para um grafo com 20 vértices. É a enormidade destes tempos que tornou este problema um dos mais famosos e não resolvidos problemas matemáticos. [2]

O problema a ser estudado, como descrito no capítulo anterior, pode ser considerado uma adaptação do problema do caixeiro viajante, ou seja, existe uma necessidade que um vértice qualquer possa ser visitado mais de uma vez, para que o caminho de volta seja o menor possível.[2]

2.2 ALGORITMO DE DIJKSTRA

O algoritmo de Dijkstra é o mais famoso dos algoritmos para cálculo de caminho de custo mínimo entre vértices de um grafo e, na prática, o mais empregado. Escolhido um vértice como raiz da busca, este algoritmo calcula o custo mínimo deste vértice para todos os demais vértices do grafo. O algoritmo pode ser usado sobre grafos orientados (dígrafos), ou não, e admite que todas as arestas possuem pesos não negativos (nulo é possível). Esta restrição é perfeitamente possível no contexto de redes de transportes, onde as arestas representam normalmente distâncias ou tempos médios de percurso; poderão existir, no entanto, aplicações onde as arestas apresentam pesos negativos, nestes casos o algoritmo não funcionará corretamente.[4]

- **FUNCIONAMENTO DO ALGORITMO**

Assumiremos um conjunto, nomeado PERM, que contém inicialmente apenas o vértice fonte (raiz da busca) **s**. A qualquer momento PERM contém todos os vértices para os quais já foram determinados os menores caminhos usando apenas vértices em PERM a partir de **s**. Para cada vértice **z** fora de PERM matém-se a menor distância dist[z] de **s** a **z** usando caminhos onde o único vértice que não está em PERM seja **z**. É necessário também armazenar o vértice adjacente (precedente) a **z** neste caminho em path[**z**].[4]

Para fazer com que PERM cresça, ou seja, deve ser incluído em PERM o vértice, entre todos os que ainda não pertencem a PERM, com menor distância dist. Acrescentamos então este vértice - nomeado de current - a PERM, e recalculando as distâncias (dist) para todos os vértices adjacentes a ele que não estejam em PERM, pois pode haver um caminho menor a partir de **s**, passando por current, do que aquele que havia antes de current ser agregado a PERM. Se houver um caminho mais curto, deve-se atualizar path[**z**] de forma a indicar que current é o vértice adjacente a **z** pelo novo caminho mínimo.[4]

2.3 ALGORITMOS EXATOS

Os métodos exatos para o problema de roteamento produzem resultados eficientes em termos de tempo computacional quando aplicados a problemas pequenos (máximo 30 vértices).

São algoritmos em que sempre se consegue uma solução ótima para o problema a ser resolvido.

2.3.1 FORÇA BRUTA

A técnica de *força bruta* procede à exploração do espaço (ou conjunto de soluções) como a pesquisa em uma árvore que representa as decisões possíveis, e tem como folhas todas as soluções. Esta árvore não existe fisicamente, mas é gerada em tempo de execução. Para obter a solução ótima deste problema, faz-se necessário percorrer todos os

caminhos possíveis e encontrar qual deles possui o menor custo. A opção tomada é a de usar busca em profundidade (BFS) com *backtracking*, para encontrar todos os caminhos. No pior caso, se tem a exploração exponencial completa, e se houver uma solução ela será encontrada.

2.3.2 BRANCH-AND-BOUND

Segundo (BOAVENTURA,1996), os algoritmos exatos são habitualmente da classe “Branch-and-bound” (ramifica e limita). Trata-se de buscas em árvores, caracterizadas pelo particionamento do conjunto de soluções por um critério dado. Em seguida se determinam limites para os valores das soluções de cada subconjunto e se opta, a cada iteração, pelo conjunto que ofereceu o menor limite inferior até o momento.

No algoritmo exato de (YANESSE, 1997) enumera-se a ordem em que cada tipo de item é completado. Na árvore de busca do método, cada nó ramificado representa um tipo de item que foi completado.[3]

Dado um problema com *n* padrões e *m* itens diferentes, exemplifica-se o esquema de enumeração do algoritmo da seguinte forma:

Do nó inicial se ramificam os *m* possíveis tipos de itens completados por um determinado seqüênciamento de padrões. Ramifica-se, a partir do primeiro item completado, os demais itens que não foram completados ainda e assim sucessivamente até que todas as possibilidades sejam analisadas.No pior caso o algoritmo é exponencial, *cf*. (YANESSE, 1997).[3]

Em cada nó ramificado *i* identifica-se um conjunto dos itens abertos *Soi* (número de pilhas abertas) e um conjunto dos itens ainda inacabados (não completados) *Sui*.

Para percorrer a árvore de busca é utilizado um critério guloso que escolhe o menor *Soi* para a próxima verificação no nível mais elevado corrente. Com isso obtêm-se, rapidamente, uma solução para o problema. Essa solução serve como uma solução inicial para a “poda” dos demais nós ainda não pesquisados. Um nó cujo *Soi* é maior ou igual que a solução obtida não precisa ser pesquisado, pois já se possui uma solução melhor.

Uma desvantagem desse método é que, no pior caso, todas as configurações são consultadas, o que faz com que o algoritmo tenha complexidade exponencial no pior caso, tornando impraticável para conjuntos de características grandes. Por essas razões existem os

métodos sub-ótimos, os quais não garantem que o conjunto de características obtido seja o melhor possível, mas são eficientes em termos de tempo de execução, pois eles não consultam todas as possibilidades para determinar a(s) solução(ões).[3]

2.4 HEURÍSTICAS

As heurísticas foram consideradas durante muito tempo modelos cognitivos por excelência, elas constituem-se como regras baseadas na experiência e no planejamento substituindo as anteriores baseadas na procura algorítmica que chega às soluções corretas depois de ter combinado o problema com todas as soluções possíveis.

Os métodos heurísticos procuram um grau tão grande quanto possível de uma ação a uma situação. Assim ela engloba estratégias, procedimentos, métodos de aproximação tentativa/erro, sempre na procura da melhor forma de chegar a um determinado fim. Os processos heurísticos exigem muitas vezes menos tempo que os processos algorítmicos, aproximam-se mais da forma como o ser humano raciocina e chega às decisões acertadamente e garantem soluções eficientes.

2.4.1 VIZINHO MAIS PRÓXIMO

Vizinho mais próximo (VMP), é uma heurística, que percorre os vértices, comparando com os seus vizinhos, qual dele está mais próximo, até completar todo o caminho. No capítulo seguinte será demonstrada sua implementação.

2.4.2 ALGORITMO GENÉTICO

Os algoritmos genéticos (AG’s) são procedimentos de busca que têm sido usados na solução dos mais variados problemas de diferentes domínios, como programação automática, ecologia, pesquisa operacional e etc. (GARCIA, 2000). Para (REBELLO e HUMACHRE, 2000), os AG´s são métodos robustos, que podem ser utilizados para resolver problemas em pesquisa numérica, otimização de funções e aprendizagem de máquina, entre

outras aplicações. Eles têm sido apontados como uma técnica promissora para resolver problemas combinatoriais. Apesar de não garantirem otimalidade, sua grande vantagem está no melhor desempenho ou performance computacional, além de ser um procedimento de relativa simplicidade. Segundo (GARCIA, 2000), praticamente todos os problemas NP-completos (complexidade exponencial) têm uma versão de solução heurística que usa esse tipo de algoritmo.

De acordo com(GARCIA, 2000), os AG`s foram desenvolvidos por John Holland e seu grupo de pesquisas, mas a teoria sobre algoritmos genéticos só foi formalizada em 1975 com o aparecimento do livro Adaptation in Natural and artificial systems de John Holland. Posteriormente (GOLDBARG, 2000) aperfeiçoou a idéias desenvolvidas por Holland.

Os AG`s são métodos de busca e otimização baseados nos princípios de seleção natural e reprodução genética. Eles empregam um processo adaptativo e paralelo de busca de soluções em problemas complexos e se enquadram na classe de métodos heurísticos inteligentes ou Metaheurísticas (REBELLO e HUMACHRE, 2000).

2.4.2.1 Princípios básicos

O algoritmo genético inicia o processo de otimização a partir de um conjunto de configurações (população inicial) que pode ser obtida aleatoriamente ou usando algoritmos heurísticos construtivos simples e rápidos (ROMERO e GALLEGO, 2000).

Em cada iteração é obtido um novo conjunto de configurações (nova população) a partir da população corrente usando os operadores de seleção, crossover e mutação. Em cada nova iteração são encontradas configurações de melhor qualidade e, eventualmente, nesse processo iterativo pode ser encontrada a solução (configuração) ótima global. Segundo (ROMERO e GALLEGO, 2000), um algoritmo genético realiza uma busca usando um conjunto de soluções (configurações) e através de um processo iterativo são encontradas novas configurações candidatas. Ainda, de acordo com (ROMERO e GALLEGO, 2000)o número de configurações visitadas nesse processo de busca deve ser um número muito reduzido de configurações do espaço de configurações e deve existir uma estratégia adequada para visitar as configurações mais atrativas.

Os princípios básicos de um AG, segundo (GOLDBARG, 2000) , pode ser resumido através do seguinte pseudocódigo:

```
Algoritmo Genético

Início
 Gerar uma população inicial
 Avaliar a fitness dos indivíduos da população
      Repetir
       Início
       Selecionar um conjunto de pais na população
       Cruzar os pais de modo que se reproduzam
       Avaliar a fitness dos filhos gerados
       Substituir os filhos julgados inadequados
       Fim
       Até quando critério de parada seja atendido
 Fim
```

2.4.2.2 Elementos do Algoritmo Genético

Independente da sofisticação de um AG, cinco componentes que são básicos em qualquer implementação de AG (IGNÁCIO, 2000):

1. Representação das soluções do problema em termos de cromossomos.

2. Modo de criar a configuração da população inicial.

3. Uma função de avaliação (aptidão) que permita ordenar os cromossomos, de acordo com a função objetivo.

4. Operadores genéticos que permitam alterar a composição dos novos cromossomos gerados pelos pais, durante a reprodução.

5. Valores dos parâmetros que o AG usa (tamanho da população; probabilidades associadas com a aplicação dos operadores genéticos: critério de seleção, critério de sobrevivência dos cromossomos; taxa de mutação; critérios de parada; etc.)

A escolha de muitos destes componentes é um processo empírico e depende da experiência e do sentimento implementador (REBELLO e HUMACHRE,2000).

2.4.2.3 Codificação das soluções do problema em termos de cromossomos

Na terminologia dos AG's um cromossomo representa um indivíduo na população (i.e. uma configuração ou solução), definido normalmente como um vetor de componentes (GOLDBARG, 2000) . Cada cromossomo apresenta os seguintes elementos: genes, alelo, lócus e fitness (Figura 2). Os genes (G) representam as componentes do cromossomo (i.e. uma variável do problema). Os alelos descrevem os possíveis estados de um atributo do indivíduo (i.e. os possíveis valores de uma variável do problema). Em problemas com variáveis binárias tem-se G E {0,1}. Finalmente, os lócus representam a N-ésima posição do atributo no cromossomo (i.e. a posição da componente no vetor dos componentes). A Fitness é a medida da aptidão do indivíduo (cromossomo). Normalmente associada ao valor da função objetivo para uma dada solução.

Figura 2 - Exemplo da estrutura de um cromossomo:

=

3

0

A representação desses cromossomos é apontada pelos especialistas como um fator determinante no sucesso do AG. A forma de codificação mais difundida é a codificação binária. Nesta codificação, cada cromossomo é representado por uma seqüência inteira de bits (Figura 4).

Figura 3 - Esquema de codificação binária de um cromossomo:

1	0	0	1	1	0	1	0

Segundo (MITCHELL,1996), a maior difusão dessa codificação pode ser atribuída a fatores históricos. Os trabalhos originais de Holland e seus colaboradores concentravam-se nessa forma de codificação e as aplicações de AG tenderam a seguir essa idéia, uma vez que muitos fundamentos do método como: Crossover, mutação e outros parâmetros do algoritmo são baseados nessa codificação. Segundo o mesmo autor, apesar dos fundamentos básicos do AG poderem ser estendias para outras formas de codificações, nem sempre apresentam boas performances, e às vezes não é uma tarefa muito fácil.

Entretanto, em muitas situações a codificação binária é inadequada para representar a solução do problema, como na representação de pesos em redes neurais, seqüências de proteínas, rotas de veículos etc. Portanto, outras formas de codificação devem ser utilizadas, para tais propósitos, como a codificação alfabética e a codificação real. Na codificação alfabética, os alelos são representados por uma seqüência de caracteres alfabéticos, enquanto na codificação real, tais alelos são representados por seqüências de números reais.

Figura 4 - Esquema da codificação real e alfabética de um cromossomo:

Codificação alfabética

A	C	A	T	U	C	A	U

Codificação real

4	8	0,5	7	1	3	4	8

Segundo (MITCHELL,1996), a melhor codificação depende muito mais do problema e os detalhes utilizados no AG, e não há nenhuma forma eficiente de prever qual codificação é mais adequada. Portanto, a experimentação é a melhor forma para comparar o mecanismo de codificação mais adequado.

2.4.2.4 População inicial

O algoritmo genético inicia o processo de otimização a partir de um conjunto de configurações (população inicial). A população em AG refere-se a um conjunto de indivíduos ou um conjunto de soluções do problema (GOLDBARG, 2000) . Em relação à população existem dois aspectos que devem ser especificados. O tamanho da população N e a forma em que é determinada a população inicial. O tamanho da população deve ser escolhido levando em conta o tamanho do problema e os outros parâmetros genéticos, como a taxa de Crossover e a taxa de mutação (ROMERO e GALLEGO, 2000).

A forma de gerar uma população inicial pode ser feita de duas maneiras diferentes: de forma aleatória ou utilizando heurísticas (IGNÁCIO, 2000). Ao se utilizar uma heurística como população inicial, pode-se aprender muito sobre as características do problema e, sobretudo, da evolução deste tipo de população pode-se obter uma configuração satisfatória para o problema de otimização (IGNÁCIO, 2000).

Durante o processo evolutivo, esta população é avaliada. A próxima geração será uma evolução da anterior e, para que isto ocorra, os mais aptos deverão possuir maior probabilidade de serem selecionados para dar origem à nova geração, que deverá ser melhor que a que lhe deu origem. Uma geração mais apta ao ambiente significa uma melhora no valor da função objetivo. Em cada “passo evolutivo” conhecido como geração; decodifica-se um indivíduo e avalia-se sua capacidade “reprodutiva”. A partir de então, cria-se uma nova população, por processos de seleção e posterior “cruzamento” de cromossomos da população anterior (GARCIA, 2000).

2.4.2.5 Seleção

Após decidir sobre a codificação a ser adotada e a geração da população inicial ter sido efetuada para o algoritmo genético, a decisão seguinte refere-se à forma de executar a seleção, isto é, como escolher os indivíduos “pais” na população que gerará os novos descendentes ou “filhos” para próxima geração.

Uma função de aptidão (fitness) é utilizada para quantificar a qualidade genética dos Cromossomos-pais, e geralmente corresponde à função de custo em problemas de otimização combinatória (IGNÁCIO, 2000). De acordo com (IGNÁCIO, 2000), esta função

de aptidão também é utilizada para decidir se um cromossomo gerado, através de um crossover, substitui ou não um cromossomo reprodutor. O propósito de escolher cromossomos-pais para o cruzamento é incrementar a probabilidade de reproduzir elementos da população que tenham bons valores na função objetivo. Um indivíduo com um valor de aptidão alto possui uma maior probabilidade de contribuir com um ou mais filhos na próxima geração.

Segundo (MITCHELL,1996), seleção muito forte reduz a diversidade da população necessária para sua evolução, ocorrendo a chamada “convergência prematura” do algoritmo, por outro lado, seleção muito fraca resultará em evolução muito lenta da população, o que significa uma convergência lenta do algoritmo.

De acordo com (GARCIA, 2000), no processo de otimização quando se implementa o algoritmo genético, a seleção é a etapa mais crítica, pois é a que consome mais tempo com cálculos da função de aptidão e probabilidade de sobrevivência de cada indivíduo. Desta forma, quanto maior o número de indivíduos, maior tempo será gasto no processo de seleção, e conseqüentemente, a velocidade de convergência do algoritmo genético está relacionada diretamente como tamanho da população (GARCIA, 2000). Ainda, segundo o mesmo autor, grande parte do custo computacional do algoritmo está concentrada na fase de seleção.

Diversos esquemas de seleção têm sido propostos na literatura de AG. Isso é ainda uma questão aberta para AG’s, sendo que a experiência e os ensaios são ainda a melhor forma para selecionar um dado método (MITCHELL,1996).

2.4.2.6 Método de seleção

Segundo (JOHNSON), citado por (MITCHELL,1996), o método de seleção proporcional é um dos muitos métodos de seleção que força o AG a reter alguns dos melhores indivíduos em cada geração. Tais indivíduos poderiam ser perdidos se eles forem submetidos a crossover ou mutação. Muitos pesquisadores argumentam que método de seleção pode melhorar significativamente a performance do algoritmo genético (MITCHELL,1996).

- **Seleção proporcional**

Na seleção proporcional, cada indivíduo i possui uma expectativa de sobrevivência E_i associada ao seu valor de aptidão, geralmente dada por:

$$E_i = fi \,.\, \frac{fi}{\sum_{i=1}^{n} fi}$$

Em que fi é a função objetivo avaliada no indivíduo i.

Uma vez que fi deve ser sempre positiva, se a função objetivo for negativa para algum problema específico, o que pode ocorrer normalmente, se constrói um operador $\psi : f \rightarrow f'$ seja sempre positiva. A função *f'* é denominada função de aptidão.

A operação de reprodução ou seleção é um processo em que se copia indivíduo de acordo com seus valores de aptidão dados uma função objetivo *f*. O operador de seleção pode ser implementado de diversas maneiras, sendo considerada a mais fácil a que é denominada "roleta russa", conforme mostrado por(GARCIA, 2000).

O método da roleta pode ser implementado da seguinte forma (MITCHELL,1996): cada indivíduo é assinalado numa roleta com uma fatia proporcional ao seu fitness. A roleta é girada N vezes, onde N é o número de indivíduos da população. Em cada giro, o indivíduo sobre a marca da roleta é selecionado para estar no grupo dos pais da próxima geração.

Ao se utilizar à seleção proporcional, deve-se evitar a existência de "super indivíduo" no início do processo, proporcionado por cromossomos que possuem probabilidades de sobrevivência é muito maior do que a média da população (MITCHELL,1996). A existência desses indivíduos é indesejável, uma vez que eles podem convergir para uma solução logo no início do processo. Essa situação recebe o nome de "convergência prematura".

(GARCIA, 2000) propõe uma forma de evitar esse problema utilizando a transformação linear de F dada por:

F(*f*) = *f*’ = a*f* +b, que deve satisfazer as seguintes condições:

i) O valor médio de *f*´ deve ser igual ao valor médio de *f*.

ii) O maior valor de *f*’ deve ser no máximo um múltiplo pré-estabelecido de média de *f*.

- **Seleção por escalonamento**

A seleção por escalonamento utiliza a função “sigma scalling” para calcular a expectativa de sobrevivência dos indivíduos.

Segundo (MITCHELL,1996), a vantagem desse método é evitar a convergência prematura mantendo a pressão de seleção constante durante toda a busca. Como no início da corrida, o desvio padrão da população é geralmente maior, haverá uma penalização para os indivíduos que apresentarem fitness muito acima da média tendendo a reduzir as diferenças de probabilidade de sobrevivência entre indivíduos mais aptos e menos aptos. Igualmente, no fim da corrida, onde geralmente reduz-se o desvio padrão da população, os indivíduos mais aptos terão maior chance de sobrevivência, permitindo a evolução continuar.

- **Seleção de Boltzman**

A seleção por escalonamento mantém a pressão de seleção constante ao longo da corrida. Entretanto, segundo (MITCHELL, 1996), freqüentemente a variação na pressão de seleção ao longo da corrida pode ser uma importante estratégia para melhorar a performance do AG.

Uma forma de implementar essa estratégia é utilizar a “seleção de Boltzmann”, segundo (MITCHELL,1996), uma abordagem similar a simulated annealing, na qual a “temperatura” controla continuamente a pressão de seleção. A temperatura inicia alta, o que significa que a pressão de seleção é baixa (i.e, cada indivíduo tem uma probabilidade razoável de reproduzir). A temperatura é diminuída gradualmente, de acordo com o esquema

prefixado, o que gradualmente aumenta a pressão de seleção, assim permitindo o AG intensificar a busca em regiões mais promissoras, propiciada pela maior probabilidade de sobrevivência dos indivíduos mais aptos.

- **Seleção por Ranqueamento**

Seleção por ranqueamento é um método cujo propósito é também prevenir convergência muito rápida (MITCHELL,1996). Segundo esse mesmo autor, os indivíduos na população são ranqueados de acordo com o seu fitness, e a probabilidade de sobrevivência de cada indivíduo depende de seu rank ao invés de seu fitness absoluto. As vantagens deste método, segundo (MITCHELL,1996) é que ele diminui a pressão de seleção permitindo que indivíduos menos aptos possam reproduzir, uma vez que os valores absolutos do fitness não são considerados, e sim a sua posição no rank. Como desvantagens, pode haver problemas de convergência do algoritmo, com uma baixa pressão de seleção. Outra desvantagem apontada é grande custo computacional necessário para classificar (ranquear) os indivíduos em cada geração.

- **Seleção Tournament**

Neste método, também denominado seleção por jogos, dois indivíduos são escolhidos ao acaso na população. Escolhe-se um parâmetro K. em seguida, um número aleatório r é então escolhido entre 0 e 1. se $r < k$, os dois indivíduos são selecionados para serem pais; caso contrário os dois são então recolocados na população inicial e podem ser selecionados novamente.

Segundo MITCHELL, a vantagem deste método, além de sua simplicidade, é a economia no processamento de operações como cálculo de funções fitness, cálculo de médias, desvio padrão, classificação de indivíduos etc, típicos dos outros métodos de seleção. Com isso, um menor esforço computacional é requerido. Entretanto, por requerer poucas informações do sistema pode aprender pouco sobre as suas características, resultando em baixa performance do algoritmo em termos da qualidade da solução obtida.

2.4.2.7 Operadores genéticos

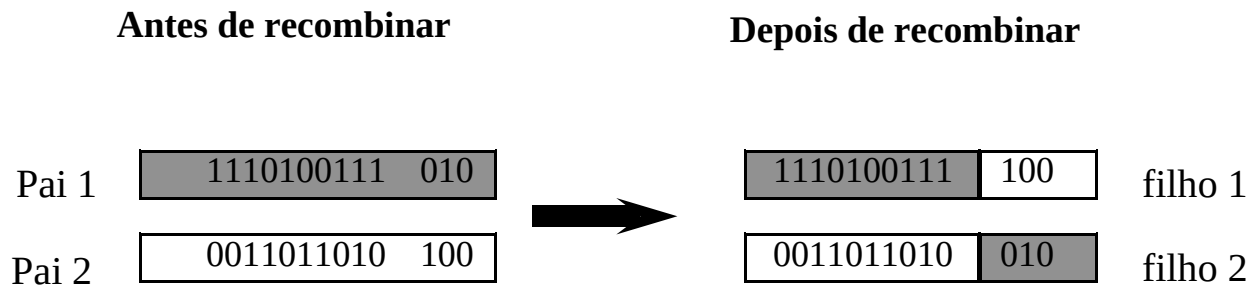
Operadores Genéticos: são as regras que permitem a manipulação dos cromossomos que, basicamente são (GOLDBARG, 2000) : Cruzamento (crossover) e mutação.

- Crossover

Este operador permite a obtenção de indivíduos filhos a partir da combinação (cruzamento) dos cromossomos dos pais (GOLDBARG, 2000) . O procedimento é simples, escolhem-se os cromossomos pais, agrupa-se em dois e cruza-os para formar dois cromossomos filhos, que terão grande probabilidade de ter uma melhor imagem quando aplicados a função objetivo. O operador de crossover, juntamente com a mutação, é responsável pela evolução dos indivíduos. O cruzamento é feito na esperança de que dois pais considerados aptos como soluções transmitem aos seus filhos as suas características boas, gerando filhos melhores do que eles, e com melhores chances da combinação de características ser uma configuração ótima para o problema (REBELLO e HUMACHRE, 2000).

A operação de crossover é validada da seguinte forma. Uma população composta por N indivíduos é agrupada aleatoriamente em pares a fim de gerar um conjunto de N/2 progenitores (“Pais”) potenciais, que são escolhidos para os cruzamentos. Atribui-se a cada par de pais uma probabilidade Pc de cruzamento. Gera-se para cada par um número aleatório entre 0 e 1. Em seguida, compara-se o valor gerado com a probabilidade Pc. Se o número encontrado for inferior a Pc, o cruzamento é permitido, caso contrário, os progenitores são mantidos inalterados e passados para a próxima geração (GARCIA, 2000).

Figura 5 – Exemplo de crossover:



- Mutação

A mutação é o operador que permite a produção de um novo indivíduo por alterações aleatórias diretas no cromossomo. Segundo (IGNÁCIO, 2000), ainda que por muito tempo a mutação tenha sido concebida com um operador secundário, ela representa um aspecto importante do AG. De acordo com (GARCIA, 2000), o propósito da operação de mutação é manter a diversidade na população, evitando que ela convirja muito rapidamente para um mínimo local (convergência prematura). Segundo este mesmo autor, ela permite também que o algoritmo genético possa gerar ou recuperar informações que poderão ser valiosas. Ainda, segundo (GARCIA, 2000), o operador de mutação deve ser usado com cautela, pois uma taxa de mutação alta aumenta a possibilidade de um "bom" indivíduo ser destruído.

A mutação é uma alteração aleatória do valor de uma posição do cromossomo, ou seja, o valor de um determinado gene do cromossomo é invertido. No caso binário, a mutação consiste em substituir, com probabilidade Pm (taxa de mutação), o valor de um bit. Para outros tipos de codificação geralmente é possível definir outras alternativas de mutação (ROMERO e GALLEGO, 2000). A mutação não passa por testes de aptidão uma vez que o objetivo é introduzir diversidade na população I.

Figura 6 – Exemplo de mutação:

Antes da mutação Depois da mutação

1100101111	1100101101

O gene 9 sofreu mutação.

2.4.2.8 Parâmetros de controle de um algoritmo genético

A eficiência de um algoritmo genético é altamente dependente de seus parâmetros de controles, tais como o tamanho da população, taxa de crossover e a taxa de mutação. Existem muitos trabalhos desenvolvidos com a finalidade de dimensionar os valores dos parâmetros na literatura de computação evolucionária (MITCHELL,1996). Entretanto, não há resultados conclusivos sobre os melhores valores desses parâmetros para todas as aplicações. Muitas aplicações usam valores reportados em outros trabalhos, outras definem os seus próprios valores com base em experimentação. Segundo (GARCIA, 2000), há consenso na literatura de AG que em geral o tamanho de cada população não deve ser menor que 25, a probabilidade de cruzamento deve estar compreendida entre 0,5 e 0,95 e probabilidade de mutação ser menor que 0,01. Segundo (MITCHELL,1996), as abordagens que parecem ser muitos promissoras é ter os valores dos parâmetros adotados em tempo real continuamente na busca. Segundo o mesmo autor, tem havido diversas abordagens adaptativas dos parâmetros de AG que têm sido um grande foco de pesquisas pelas comunidades de computação evolucionária.

2.4.2.9 Formalização do Algoritmo Genético

Segundo (GOLDBARG, 2000) , um AG operacionaliza seu processo evolucionário através do controle de adequação dos indivíduos da população (cromossomos), sendo a avaliação de adequação um dos seus elementos centrais para a promoção da adaptação. Estes autores definiram formalmente um AG da seguinte forma:

$AG = (N,P,F,\Theta,\Omega\tau);$

Onde P é uma população de N indivíduos, P =(S1, S2...Sn). Cada indivíduo Si, i = 1..N é um conjunto de valores representando uma solução para o problema. F é á função Fitness ou Adequação que retorna um valor real e positivo na avaliação do indivíduo.Θ é o operador de seleção que permite escolher r indivíduos de P. Ω é o conjunto de operadores genéticos que inclui o operador de Crossover - Ωc e o operador de Mutação - Ωm. ψ é o operador de remoção que permite eliminar indivíduos da população para que novos indivíduos mais adaptados sejam adicionados. τ é o critério de parada.

3 ALGORITMO BRANCH-AND-BOUND

O algoritmo utilizado para obter a solução ótima para o problema descrito no capítulo 1, utiliza uma matriz de adjacência como estrutura de dados, gerada em um arquivo, onde ao executar o programa deve-se abrir a mesma, para realização dos testes, e com valores conhecidos para $n = < 257$ vértices, fornecidos para submissão automática do resultado obtido. A implementação se deu na linguagem Delphi e pode ser resumido nos seguintes passos:

1. Aplicar busca em profundidade no grafo dado pela matriz de distâncias;

2. Inserir o valor (-1) no vértice visitado que não possui mais vizinho ;

3. Utilizar a técnica de *backtracking* , utilizando o procedimento de *Dijkstra* forma a permitir que todos os caminhos possíveis sejam verificados;

4. Calcular o custo do caminho encontrado;

5. Cortar chamadas a *função de visita dos vértices* tão logo se chegue a um custo para qualquer caminho que seja maior que um caminho solução já obtido (técnica de poda *branch and bound*)

6. O menor custo de caminho encontrado é a solução ótima para o problema.

Neste programa, é necessário, antes de executá-lo, que se abra um arquivo contendo uma matriz com n x n números que podem ou não ser distintos, sendo que a diagonal principal da matriz possui valor zero.

Esta matriz representa os caminhos entre os n vértices que a compõem.

A partir destes dados é implementado um algoritmo que determine o menor caminho que inicia no vértice de origem, passa em todas os demais vértices e retorna ao vértice de origem.

O algoritmo utiliza busca em profundidade para encontrar um caminho, e backtracking para encontrar um novo caminho. Toda vez que este novo caminho for maior que o menor caminho já encontrado, é feita a poda na árvore de caminhamento (técnica de poda branch and bound). O menor entre os caminhos será a solução para o problema.

Os testes foram efetuados considerando a matriz de 257 vértices.

O código do algoritmo se encontra no anexo I.

2.5 ALGORITMO VIZINHO MAIS PRÓXIMO (VMP)

O algoritmo utilizado como heurística para obter uma boa solução para o problema descrito no capítulo 1, utiliza uma matriz de adjacências como estrutura de dados, cujos valores são obtidos de um arquivo contendo os caminhos para cada vértice.A implementação se deu na linguagem Delphi,e pode ser resumido nos seguintes passos:

1. Definir um vértice de origem;

2. Encontrar a menor distância entre este e os demais vértices;

3. O vértice com menor distância passa a fazer parte do caminho, e a partir dela repetir o passo 2 apenas nos vértices não visitados, até que não exista vértice não visitado;

4. Adicionar ao final do caminho o vértice de origem e encontrar o custo do caminho;

5. Repetir os passos de 1 a 4 para os demais vértices, cada uma iniciando como origem;

6. O menor custo de caminho encontrado é a solução heurística para o problema.

Este programa lê os dados em arquivo e gera uma matriz de adjacências com números que podem ser distintos, sendo que a diagonal principal da matriz possui valor zero. Esta matriz representa as distâncias entre os n vértices que a compõem. A partir destes dados é implementado um algoritmo que determine um caminho que inicia no vértice de origem, passa em todos os demais vértices uma única vez, e retorna ao vértice de origem.

O custo deste caminho deve ser o menor possível.

O algoritmo utiliza a heurística do vizinho mais próximo, com vários caminhos dados pelas origens em cada vértice. A heurística consiste em, a partir da origem, encontrar a menor distância entre o vértice i e i+1, e assim sucessivamente até visitar todos os vértices, retornado então à origem. O menor entre os caminhos será a solução para o problema.

O código do algoritmo se encontra no anexo II.

2.6 ALGORITMO GENÉTICO

A implementação se deu na linguagem Delphi, e pode ser demonstrada através do seguinte pseudocódigo:

Algoritmo Genético

Início

Gerar uma população inicial

Repetir

Início

Selecionar um conjunto de pais na população

Cruzar os pais de modo que se reproduzam

Substituir os filhos julgados inadequados

Fim

Até quando critério de parada seja atendido

Fim

A grande dificuldade encontrada na implementação foi gerar uma população inicial, que foi implementada da seguinte forma:

Foi criada uma tabela onde uma coluna é preenchida com os vértices, e as demais colunas com os sucessores e antecessores.

Exemplo:

A B C

A C B

B A C

C B A

Foram definidas as seguintes regras para geração da população inicial:

Regra 1:

Número de vértices visitados;

Regra 2:

Tempo que o vértice foi visitado;

Regra 3:

Número de bordas (sucessor e antecessor) de cada vértice;

Regra 4:

Rondon (aleatório).

Com a implementação da tabela de bordas descrita, obteve-se um resultado satisfatório. O método de seleção implementado foi o proporcional.

O código do algoritmo se encontra no anexo III.

3 RESULTADOS OBTIDOS

Foram efetuados testes com o auxílio de um computador Pentium IV 1.8 GHz com 512 MB de RAM, no qual, foi executado os algoritmos descritos para a medição de eficiência, com isso, obteve os seguintes resultados conforme tabela abaixo:

Tabela 1 – Tempo de Execução:

Vértices (n)	Branch-and-Bound	VMP	Genético
8	**<1 s**	**<1 s**	**<1 s**
11	**<1 s**	**<1 s**	**<1 s**
12	**2 s**	**<1 s**	**<1 s**
15	**3 s**	**<1 s**	**<1 s**
17	**4 s**	**<1 s**	**<1 s**
19	**1’ 02 s**	**11 s**	**<1 s**
21	**5’ 45 s**	**54 s**	**<1 s**
28	**-**	**-**	**<1 s**

Pode-se observar que o Algoritmo de Branch and Bound apresenta um melhor resultado de caminho, mas é ineficiente para cálculos acima de 28 vértices, devido a sua complexidade exponencial. O Algoritmo do Vizinho mais Próximo (VMP) ocorre uma melhora no resultado de tempo, devido o algoritmo ser uma heurística. O Algoritmo Genético apresenta um tempo satisfatório de processamento, consegue solucionar problemas reais, ou seja, acima de 200 vértices, mas não apresenta o melhor caminho, e sim um caminho aproximado do ótimo.

4 CONCLUSÃO

O problema estudado de otimização envolve a utilização de algoritmos da classe *NP*-Completo, ou seja, a cada vértice acrescentado, seu tempo de execução se transforma em exponencial, e portanto é intratável. O uso de poda ou outro tipo de otimização podem obter resultados melhores para pequenas instâncias, isto, contudo não torna o problema mais fácil de ser resolvido. A forma mais adequada, então, é utilizar heurísticas que produzam bons resultados, ainda que não sejam as soluções ótimas, em tempo polinomial no tamanho da entrada.

Utilizando heurísticas para resolver o problema, os resultados obtidos para o algoritmo VMP foram melhores que os encontrados no algoritmo Branch-and-Bound, contudo não consegue encontrar a solução para um problema real.

A utilização do algoritmo genético, obteve resultados satisfatórios quanto a solução do problema, mas é preciso deixar claro que a solução encontrada não é a melhor e sim uma boa solução.

O estudo dos algoritmos dá a seguinte conclusão sobre suas complexidades: Sendo um método exato, o algoritmo Branch and Bound retorna o melhor caminho, mas a

cada iteração, quando se aumenta o numero de vértices, observa-se que o método não é viável, tendo sua complexidade definida como exponencial.

O algoritmo Vizinho mais Próximo é definido como uma heurística, obtendo-se uma melhora razoável ao método exato, mas ainda não encontra uma solução para um problema real.

O algoritmo Genético como o VMP, se classifica como uma heurística, não retorna a melhor solução, mas em conseqüência resolve um problema real, portanto não existe uma complexidade a ser definida.

4.1 TRABALHOS FUTUROS

Fica como sugestão, uma melhora na estrutura do algoritmo genético.

Pode-se estabelecer o janelamento de tempo para fazer com que seja executado o percurso dentro de um tempo pré-definido.

REFERÊNCIAS BIBLIOGRÁFICAS

[1]-Universidade Federal de Uberlândia - Otimização - disponível em http://www.propp.ufu.br/revistaeletronica/A/OTIMIZACAO.PDF/ 20 de Março 2004

[2]-Grupo DCT – Caixeiro Viajante - disponível em http://www.dct.ual.pt/io1/Grafos_cap5.pdf / 25 de Março 2004

[3]- Universidade Federal do Rio Grande do Sul – Branch-and-Bound - disponível em http://www.inf.ufrgs.br/~johann/papers/qualify5.pdf / 26 de Março 2004

[4]- Luiz Gustavo Nonato - Algoritmo de Dijkstra – disponível em http://www.lcad.icmc.usp.br/ ~nonato/ED/Dijkstra/node84.html, 25 de junho de 2004.

(BOAVENTURA, 1996) - BOAVENTURA, PAULO OSWALDO. **Grafos: Teoria, Modelos, Algoritmos.** São Paulo, ed. Edgard Blucher, 1996, p225-273.

(GARCIA, 2000)- GARCIA, A.L.; GARCIA, B.; FRIEDMANN,. **Resolução de um problema de equilíbrio de trabalho e distribuição de agentes de serviço.** In. SIMPÓSIO

BRASILEIRO DE PESQUISA OPERACIONAL, 32, 2000, Viçosa, Anais. Viçosa: UFV, 2000. p 923-935.

(GOLDBARG, 2000) - GOLDBARG, M.C.; GOUVÊA, E.F. **Otimização combinatória e programação linear: modelos e algoritmos.** Rio de Janeiro, ed. Campos, 2000. p 649

(IGNÁCIO, 2000)- IGNÁCIO, A.J.; FERREIRA FILHO, V.J.M; GALVÃO, R.D. **Métodos heurísitocs num entorno paralelo**. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL**,** 32, 2000, Viçosa, Anais. Viçosa: UFV, 2000 p769-788.

(JOHNSON)- JOHNSON, K. N., SCHEURMAN, H. L. **Techniques for prescribing optimal timber harvest and investment under different objectives – discussion and synthesis.** Forest Science, Washington, v.18 , 1997.

(MITCHELL, 1996)- MITCHELL, M. **An introduction to genetic algorithms. London,** A Bradford Book, 1996

(REBELLO e HUMACHRE, 2000) - REBELLO, F.R.; HUMACHER, S. **Uma proposta de algoritmo de duas fases para roteamento de veículos.** In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 32, 2000, Viçosa: UFV, 2000;

(ROMERO e GALLEGO, 2000)- ROMERO, R; GALLEGO, R. A. **Algoritmo genético especializado para problemas de alocação ótima de capacitores em sistemas de distribuição de energia elétrica.** In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 32, 2000, Viçosa: UFV, 2000;

(SOUZA, 2000)- SOUZA, A.B.D.; MOCCELLIN, J.V.**Metaheurística híbrida algoritmo genético-busca tabu para programação de operações flow shop** In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 32, 2000, Viçosa: UFV, 2000;

(YANESSE, 1997) - Yanesse, C. C. F., Sant'Anna, S. J. S., Frery, A. C., Renno´, C. D., Soares, J. V., & Luckman, A. J. **Exploratory study of the relationship between tropical forest regeneration stages and SIR-C L and C data**. Remote Sensing of Environment, 59(2), 180– 190, 1997

ANEXO I – ALGORITMO BRANCH-AND-BOUND

```
Procedure AcharCaminho(Iv : Integer);
Var I : Integer ;//contador
 S : String ;
 D : Byte ;
Begin
 // Testar se achou solução...
 If (G.M[Iv,Origem] <> 0) and (Nvv = G.NV) Then //Nvv – Numero de vértices visitados
 Begin
 IncDist(Iv,Origem);
 If (BestDist = -1) or (Dist < BestDist) Then
 Begin
 S := '' ;
 For I := 0 to H do
 S := S + Caminho[I].Nome + ' -> ';
 S := S + Caminho[Origem].Nome ;
 Form1.Memo2.Lines.Add(S+FloatToStr(Dist));
 BestDist := Dist ;
 End;
 DecDist(Iv,Origem);
 End;
 For I := 0 to G.NV-1 do
 If (G.M[Iv,I] <> 0) (*and
```

```
((Dist + {(G.NV-Nvv) +} G.VDM[I] < BestDist) or (BestDist=-1)) *) (*sem poda*) Then
Begin
Inc(H);
Caminho[H] := G.V[I] ;
IncDist(Iv,I); // incrementa distância
D := G.M[Iv,I] ;
G.M[Iv,I] := 0;// grafo
If G.V[I].Vis = 0//Matriz
Then
Begin
Inc(Nvv) ;
G.V[I].Vis := 1 ;
AcharCaminho(I);
Dec(Nvv) ;
G.V[I].Vis := 0 ;
End
Else
AcharCaminho(I);
Dec(H);//distância
G.M[Iv,I] := D;
DecDist(Iv,I) ;
End;
End;
procedure TForm1.Button2Click(Sender: TObject);
Var T : TDateTime ;
begin
T := Now ;
Try
Cursor := crHourGlass ;
SetLength(Caminho,NVCaminho);
G.V[Origem].Vis := 1 ;
Caminho[H] := G.V[Origem] ;
AcharCaminho(Origem);
Finally
Edit1.Text := TimeToStr(Now-T);
Cursor := crDefault ;
End;
            end;
```

ANEXO II – ALGORITMO VMP

```
Procedure AcharCaminhoVMP(Iv : Integer);
Var I, VMP, Passo : Integer ;
 MD : Array of Array of Integer ;
 Encontrou : Boolean ;
 S : String ;
 Function EncontrarVMP(Origem: Integer) : Integer ;//VMP – vizinho mais próximo
 Var I, IMenor : Integer ;
 Function EncontrarMenor : Integer ;
 Var I : Integer ;
 Begin
 // Encontrar primeiro vértice não marcado
 I := 0 ;
 While (MD[G.NV,I]=-1) and (I < G.NV) do //vetor
 Inc(I);
 Result := I ;
 // Verificar se não existe outra opção melhor
 While (I<G.NV) do
 Begin
 If (MD[Passo,I] < MD[Passo,Result]) and
 (MD[G.NV,I]=0) Then
```

```
Result := I ;
Inc(I);
End;
End;
Begin
Result := -1 ;
// Inicialização
For I := 0 to G.NV-1 do// Número de vértices
Begin
// Abrir todos os vértices
MD[G.NV,I] := 0 ;
// Inicializar custos
MD[0,I] := Infinito ;
End;
//Inicializar custo origem
MD[0,Origem] := 0 ;
Encontrou := False ;
Passo := 0 ;
// Repetição
While not Encontrou do
Begin
// Encontrar vértice de menor custo que ainda esteja aberto
IMenor := EncontrarMenor ;
// Fechar vértice
MD[G.NV,IMenor] := -1 ;// IMenor – distância menor
MD[Passo,G.NV] := IMenor ;
For I := 0 to G.NV-1 do
If MD[Passo,IMenor] + G.M[IMenor,I] < MD[Passo,I]
Then
MD[Passo+1,I] := MD[Passo,IMenor] + G.M[IMenor,I]
Else
MD[Passo+1,I] := MD[Passo,I];
// Testar se encontrou solução
If G.V[Imenor].Vis = 0 Then
Begin
Encontrou := True ;
Result := IMenor ;
End;
Passo := Passo + 1 ;
End;
End;
Procedure IncluirVMPCaminho(VMP:Integer);
Var I, Hi, P : Integer ;
Begin
Hi := H ;
Inc(H);
```

```
Caminho[H] := G.V[VMP] ;// H – distância
P := Passo ;
While P > 0 do//Posição
Begin
If MD[P,VMP] <> MD[P-1,VMP]
Then
Begin
If P > 1 Then
Begin
For I := H Downto Hi+1 do
Caminho[I+1] := Caminho[I];
Caminho[Hi+1] := G.V[MD[P-1,G.NV]] ;
Inc(H);
VMP := MD[P-1,G.NV] ;
End;
End;
Dec(P)
End;
For I := Hi+1 To H do
Begin
IncDist(Caminho[I-1].Ind,Caminho[I].Ind);
Inc(Nvv) ;
End;
End;
Begin
SetLength(MD,G.NV+1);
For I := 0 to G.NV do
SetLength(MD[I],G.NV+1);
For I := 1 to G.NV-1 do
Begin
// Encontrar o vértice mais próximo ainda não visitado
VMP := EncontrarVMP(Iv);
// Marcar o Vértice como visitado
G.V[VMP].Vis := 1 ;
// Incluir o Caminho encontrrdo
IncluirVMPCaminho(VMP);
Iv := VMP ;
End;
// Calcular a volta
G.V[0].Vis := 0;
VMP := EncontrarVMP(Iv);
G.V[VMP].Vis := 1 ;
IncluirVMPCaminho(VMP);
S := '' ;
For I := 0 to H-1 do
S := S + Caminho[I].Nome + ' -> ';
```

```
 S := S + Caminho[Origem].Nome ;
 Form1.Memo2.Lines.Add(S+FloatToStr(Dist));
End;
procedure TForm1.Button3Click(Sender: TObject);
Var T : TDateTime ;
 I,J : Integer ;
begin
 T := Now ;
 Try
 Cursor := crHourGlass ;
 For I := 0 to G.NV-1 do
 For J := 0 to G.NV-1 do
 If G.M[I,J] = 0 Then
 G.M[I,J] := Infinito ;
 SetLength(Caminho,NVCaminho);
 G.V[Origem].Vis := 1 ;
 Caminho[H] := G.V[Origem] ;
 AcharCaminhoVMP(Origem);
 Finally
 Edit1.Text := TimeToStr(Now-T);
 Cursor := crDefault ;
 End;
end;
```

ANEXO III – ALGORITMO GENÉTICO

```
Procedure IncluirVertTabBordas(Var TabBordas : TTabBordas;I,J : Integer);
 Var K : Integer ;
 Begin
 K := 0 ;
 While (K < Mv) and (TabBordas[I,K] <> -1) do
 Begin
 If TabBordas[I,K] = J Then K := Mv-1 ;
 Inc(K) ;
 End;
 If K < Mv Then
 Begin
 Inc(G.V[I].NBordas);
 TabBordas[I,K] := J ;
 End;
 End;
 Function Cruzamento (Pai1, Pai2 : TIndividuo ) : Tindividuo ;
Var I, J, K, iVe, iVa : Integer ;
 TabBordas : TTabBordas ;
 Filho : TIndividuo ;
begin
 //// Inicializar Tabela de bordas
 SetLength(TabBordas,G.NV);
```

```
For I := 0 to G.NV-1 do
Begin
// Inicializar valores para vértices
G.V[I].NBordas := 0 ;
G.V[I].Vis := 0 ;
G.V[I].TV := 0 ;
SetLength(TabBordas[I],Mv);
For K := 0 to MV-1 do
TabBordas[I,K] := -1 ;
// Incluir vértices adjacentes
For J := 0 to Pai1.NVv-1 do
If Pai1.Caminho[J].Ind = I Then
Begin
// Sucessor
If J < Pai1.NVv-1 Then
IncluirVertTabBordas(TabBordas,I,Pai1.Caminho[J+1].Ind);
// Antecessor
If J > 0 Then
IncluirVertTabBordas(TabBordas,I,Pai1.Caminho[J-1].Ind);
End;
For J := 0 to Pai2.NVv-1 do
If Pai2.Caminho[J].Ind = I Then
Begin
// Sucessor
If J < Pai2.NVv-1 Then
IncluirVertTabBordas(TabBordas,I,Pai2.Caminho[J+1].Ind);
// Antecessor
If J > 0 Then
IncluirVertTabBordas(TabBordas,I,Pai2.Caminho[J-1].Ind);
End;
End;
/// fim da criação da tabela de bordas
/// Inicializar mutação ...
For I := 0 to G.NV-1 do
Begin
For J := 0 to G.NV-1 do
If (G.M[I,J] <> 0) and (Random < TxMutacao) Then
Begin
IncluirVertTabBordas(TabBordas,I,J);
IncluirVertTabBordas(TabBordas,J,I);
End;
End;
SetLength(Filho.Caminho,NVCaminho);
Filho.NVv := 1 ;
Filho.Caminho[0] := Pai1.Caminho[0] ;
Inc(G.V[0].Vis);
```

```
 Inc(Tempo);
 G.V[0].TV := Tempo ;

 iVa := Filho.Caminho[0].Ind ;

 Repeat
 // Escolher um vizinho
 iVe := TabBordas[iVa,0] ;
 J := 1 ;
 While (J < Mv) and (TabBordas[iVa,J] <> -1) do
 Begin
 // Primeiro Criterio
 If G.V[iVe].Vis > G.V[TabBordas[iVa,J]].Vis
 Then iVe := TabBordas[iVa,J]
 Else If G.V[iVe].Vis = G.V[TabBordas[iVa,J]].Vis Then
 // Segundo Criterio
 If G.V[iVe].TV > G.V[TabBordas[iVa,J]].TV
 Then iVe := TabBordas[iVa,J]
 Else If G.V[iVe].TV = G.V[TabBordas[iVa,J]].TV Then
 // Terceiro Criterio
 If G.V[iVe].NBordas > G.V[TabBordas[iVa,J]].NBordas
 Then iVe := TabBordas[iVa,J]
 Else If G.V[iVe].NBordas = G.V[TabBordas[iVa,J]].NBordas Then
 // Quarto Criterio (Random)
 If Random < 0.5 Then iVe := TabBordas[iVa,J] ;
 Inc(J);
 End;
 Filho.Caminho[Filho.Nvv] := G.V[iVe] ;
 Inc(G.V[iVe].Vis);
 Inc(Tempo);
 G.V[iVe].TV := Tempo ;
 Inc(Filho.Nvv);
 iVa := iVe ;
 Until iVe = Filho.Caminho[0].Ind ;
 Result := Filho ;
end;
/// Gerar individuo Aleatório
Function IndAleatorio : Tindividuo ;
Var I, J, K : Integer ;
 TabBordas : TTabBordas ;
 Filho : TIndividuo ;
begin
 Try
 //// Inicializar Tabela de bordas
 SetLength(TabBordas,G.NV);
 For I := 0 to G.NV-1 do
```

```
Begin
// Inicializar valores para vértices
G.V[I].NBordas := 0 ;
G.V[I].Vis := 0 ;
G.V[I].TV := 0 ;
SetLength(TabBordas[I],Mv);
For K := 0 to 9 do
TabBordas[I,K] := -1 ;
End;
/// Inicializar mutação ...
For I := 0 to G.NV-1 do
Begin
For J := 0 to G.NV-1 do
If (G.M[I,J] <> 0) and ( (TabBordas[I,0] = -1) OR (Random < TxMutacao)) Then
IncluirVertTabBordas(TabBordas,I,J);
End;
SetLength(Filho.Caminho,NVCaminho);
Filho.NVv := 1 ;
Filho.Caminho[0] := G.V[Origem] ;
Inc(G.V[0].Vis);
Inc(Tempo);
G.V[0].TV := Tempo ;
I := Filho.Caminho[0].Ind ;
Repeat
// Escolher um vizinho
K := TabBordas[I,0] ;
J := 1 ;
While (J < Mv) and (TabBordas[I,J] <> -1) do
Begin
// Primeiro Criterio
If G.V[K].Vis > G.V[TabBordas[I,J]].Vis
Then K := TabBordas[I,J]
Else If G.V[K].Vis = G.V[TabBordas[I,J]].Vis Then
// Segundo Criterio
If G.V[K].TV > G.V[TabBordas[I,J]].TV
Then K := TabBordas[I,J]
Else If G.V[K].TV = G.V[TabBordas[I,J]].TV Then
// Terceiro Criterio
If G.V[K].NBordas > G.V[TabBordas[I,J]].NBordas
Then K := TabBordas[I,J]
Else If G.V[K].NBordas = G.V[TabBordas[I,J]].NBordas Then
// Quarto Criterio (Random)
If Random < 0.5 Then K := TabBordas[I,J] ;
Inc(J);
End;
Filho.Caminho[Filho.Nvv] := G.V[K] ;
```

```
 Inc(G.V[K].Vis);
 Inc(Tempo);
 G.V[K].TV := Tempo ;
 Inc(Filho.Nvv);
 I := K ;
 Until K = Filho.Caminho[0].Ind ;
 Result := Filho ;
 Except
 on exception do Result.NVv := -1 ;
 End;
end;
procedure TForm1.Button4Click(Sender: TObject);
Var I : Integer ;
 Geracao : Integer ;
 NovaPopulacao : Array of TIndividuo ;
 MedPop : Extended ;
 Procedure CalcParametrosSelecao ;
 Var I : Integer ;
 FT, FTi, Acc : Extended ;
 Begin
 FT := 0 ;
 For I := 0 to TamPopulacao-1 do
 FT := FT + Populacao[I].Fitness ;
 FTi := 0 ;
 For I := 0 to TamPopulacao-1 do
 FTi := FTi + FT/Populacao[I].Fitness ;
 Acc := 0 ;
 For I := 0 to TamPopulacao-1 do
 Begin
 Populacao[I].ES := Acc + ((FT / Populacao[I].Fitness) / FTi );
 Acc := Populacao[I].ES ;
 End;
 //Memo2.Lines.Add(FloatTostr(Populacao[TamPopulacao-1].ES));
 End;
 Function SelecionarPai : Tindividuo ;
 Var R : Extended ;
 I : Integer ;
 Begin
 R := Random ;
 For I := 0 to TamPopulacao-1 do
 If R <= Populacao[I].ES Then
 Begin
 Result := Populacao[I] ;
 Exit ;
 End;
 End;
```

```
Procedure MostrarPopulacao ;
Var I, J : Integer ;
S : String ;
BestFitness : Integer ;
Begin
MedPop := 0 ;
BestFitness := 0 ;
For I := 0 to TamPopulacao-1 do
Begin
Memo2.Lines.Add(intToStr(Populacao[I].Nvv));
S := '' ;
For J := 0 to Populacao[I].Nvv-1 do
S := S + Populacao[I].Caminho[J].Nome + ' -> ';
S := S + FloatToStr(Populacao[I].Dist);
S := S + '(' + FloatToStr(Populacao[I].Fitness) +')';
Form1.Memo2.Lines.Add(S);
MedPop := MedPop + Populacao[i].Fitness ;
If Populacao[Bestfitness].Fitness > Populacao[i].Fitness Then
BestFitness := I ;
End;
MedPop := MedPop / Tampopulacao ;
S := 'Media : (' + FloatToStr(medPop) +')';
S := FloatToStr(Populacao[BestFitness].Fitness) ;
Form1.Memo2.Lines.Add(S);
// mostrar o melhor
(*S := '' ;
For J := 0 to Populacao[BestFitness].Nvv-1 do
S := S + Populacao[BestFitness].Caminho[J].Nome + ' -> ';
S := S + FloatToStr(Populacao[BestFitness].Dist);
S := S + '(' + FloatToStr(Populacao[BestFitness].Fitness) +')';
Form1.Memo2.Lines.Add(S);
*)
End;
begin
Randomize ;
// Construir População Inicial;
SetLength(Populacao,TamPopulacao);
SetLength(NovaPopulacao,TamPopulacao);
For I := 0 to TamPopulacao-1 do
Repeat
Populacao[I] := IndAleatorio ;
If Populacao[I].NVv <> -1 Then
CalcFitnessIndividuo(Populacao[I]);
```

```
  Until (Populacao[I].Nvv <> -1) and (Populacao[I].Dist / Populacao[I].Fitness >=
TxAceitacao);

 Geracao := 1 ;
 MostrarPopulacao ;
 Repeat
 CalcParametrosSelecao ;
 // Refazer a população
 For I := 0 to TamPopulacao-1 do
 Repeat
 Pai1 := SelecionarPai ;
 Pai2 := SelecionarPai ;
 NovaPopulacao[I] := Cruzamento(Pai1,Pai2) ;
 CalcFitnessIndividuo(NovaPopulacao[I]);
 Until NovaPopulacao[I].Dist / NovaPopulacao[I].Fitness >= TxAceitacao;
 For I := 0 to TamPopulacao-1 do
 Populacao[I] := NovaPopulacao[I] ;
 Inc(Geracao);
 MostrarPopulacao ;
 Until Geracao = Geracoes ;
end;
Procedure CalcFitnessIndividuo (Var Ind : Tindividuo);
Var I : Integer ;
Begin
 Ind.Dist := 0 ;
 For I := 0 to G.NV-1 do
 G.V[I].Vis := 1 ;
 For I := 1 to Ind.NVv-1 do
 Begin
 Ind.Dist := Ind.Dist +
 G.M[Ind.Caminho[I-1].Ind,Ind.Caminho[I].Ind] ;
 G.V[Ind.Caminho[I-1].Ind].Vis := 0 ;
 End;
 Ind.Fitness := Ind.Dist ;
 For I := 0 to G.NV-1 do
 Ind.Fitness := Ind.Fitness + G.V[I].Vis*TxPenalizacao ;
End;
```