SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 22.02.2002

Assinatura:

Uma avaliação de meta-heurísticas para o problema de designação quadrática

Thiago D'Martin Maia

Orientadora: *Profª. Dra. Regina Esther Berretú*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP – São Carlos
Fevereiro/2002

A Comissão Julgadora:

Profa. Dra. Regina Esther Berretta

Prof. Dr. Marcos Nereu Arenales

Profa. Dra. Débora Pretti Ronconi

Para Regina, que me considerou factível e procurou melhorar-me.

À minha família, com admiração;

Aos meus amigos todos;

A Marcos Nereu Arenales, Maria das Graças Volpe Nunes, Pablo Alberto Moscato, Reinaldo Morabito Neto, Nair Maria Maia de Abreu, Vitória Maria Miranda Pureza, Roseli Aparecida Francelin Romero, Edna Maura Zuffi, Renata Pontin de Mattos Fortes, Luiz Fernando Rodrigues e Luciana Salete Buriol, que me permitiram espiar seu conhecimento vasto;

Aos funcionários da USP;

À agência CAPES, que financiou o projeto;

Ao Estado de São Paulo; e

A João Ubaldo Ribeiro, pela cessão da epígrafe.

Sabe-se muito pouco.

João Ubaldo Ribeiro, em O Feitiço da Ilha do Pavão

RESUMO

Este trabalho apresenta uma avaliação de meta-heurísticas para a resolução do Problema de Designação Quadrática (*Quadratic Assignment Problem*).

O Problema de Designação Quadrática consiste em posicionar-se um determinado conjunto de instalações em um determinado conjunto de localidades, dados os fluxos entre as instalações e as distâncias entre as localidades, com o objetivo de minimizar um somatório que vincula os fluxos às distâncias de acordo com o posicionamento.

Foram implementadas heurísticas que utilizam Busca Tabu, Algoritmos Genéticos e Algoritmos Meméticos, e uma avaliação com exemplos numéricos encontrados na literatura foi realizada.

ABSTRACT

This work presents an evaluation of metaheuristics for the Quadratic Assignment Problem. The Quadratic Assignment Problem consists of assigning a set of facilities (with given flows between them) to a set of locations (with given distances between them), in such a way that the sum of the product between flows and distances is minimized.

Heuristics using Tabu Search, Genetic Algorithms and Memetic Algorithms have been implemented, and it was done an evaluation with instances found in literature.

CONTEÚDO

1. INTRODUÇÃO.....	9
2. PROBLEMA DE DESIGNAÇÃO QUADRÁTICA – QAP	11
2.1. INTRODUÇÃO	12
2.2. MODELAGENS DO QAP.....	12
2.2.1. Formulação proposta por Koopmans e Beckmann, 1957	12
2.2.2. Uma Formulação por Programação Inteira.....	13
2.3. UMA ILUSTRAÇÃO DO QAP	14
2.4. RESOLUÇÃO DO QAP.....	16
3. META-HEURÍSTICAS.....	18
3.1. INTRODUÇÃO	19
3.2. PROBLEMAS DE OTIMIZAÇÃO	19
3.3. ÓTIMOS LOCAIS E GLOBAIS.....	20
3.4. HEURÍSTICAS	20
3.4.1. As classes de problemas P e NP.....	22
3.4.2. Tipos de Heurísticas.....	23
3.5. META-HEURÍSTICAS	24
3.5.1. Busca Tabu	24
• Memória de curto prazo.....	26
• Memória de longo prazo.....	26
• Critério de aspiração	27
• Um Pseudo-Código de Busca Tabu	27
3.5.2. Algoritmos Genéticos	28
• Alguns Tipos de Operador de Cruzamento.....	30
• Um Pseudo-Código de um Algoritmo Genético	32
3.5.3. Algoritmos Meméticos	32
• Um Pseudo-Código de um Algoritmo Memético com busca local.....	34
4. META-HEURÍSTICAS IMPLEMENTADAS	35
4.1. INTRODUÇÃO	36
4.2. BUSCA TABU	36
• Representação de uma solução	36
• Movimento.....	36
• Atributo.....	37
• Critério tabu	37
• Tempo tabu	37
• Critério de Aspiração	37
• Pseudocódigo da Busca Tabu	37
4.3. ALGORITMO GENÉTICO	39
• Representação	39
• Seleção dos pais.....	39
• Operador de Cruzamento CX.....	39
• Operador de Mutação.....	41
• Função de Aptidão	42
• Atualização da População.....	42
• Pseudocódigo do Algoritmo Genético	42
4.4. ALGORITMO MEMÉTICO	43
• Pseudocódigo do Algoritmo Memético	44

5. RESULTADOS COMPUTACIONAIS.....	46
5.1. INTRODUÇÃO.....	47
5.2. BUSCA TABU	48
5.3. ALGORITMOS GENÉTICOS.....	49
5.4. ALGORITMOS MEMÉTICOS	51
5.5. COMPARAÇÃO ENTRE AS META-HEURÍSTICAS	53
6. CONCLUSÃO.....	55
7. REFERÊNCIAS BIBLIOGRÁFICAS	57

1. INTRODUÇÃO

Esse trabalho apresenta uma avaliação de algumas técnicas meta-heurísticas para a resolução do Problema de Designação Quadrática (QAP).

O QAP pode ser definido, sucintamente, como a necessidade de se otimizar a designação de um certo conjunto de instalações, entre as quais são dados os fluxos, a um certo conjunto de localidades, entre as quais são dadas as distâncias.

Para tal problema ainda não se conhece um algoritmo eficiente que apresente resolução exata. Essa dificuldade o classifica como NP-difícil.

Como principal objetivo do trabalho, serão avaliadas algumas técnicas computacionais chamadas Meta-heurísticas na resolução do QAP. Estas vêm sendo largamente pesquisadas e utilizadas com sucesso na resolução de problemas classificados como difíceis em Otimização.

O texto é organizado da seguinte forma: após esta introdução há uma apresentação detalhada do QAP, usando-se definições clássicas encontradas na literatura, exemplos gráficos e a descrição de caminhos considerados promissores na resolução desse problema. O terceiro capítulo introduz o conceito de Meta-heurísticas, algoritmos que foram avaliados para a resolução do QAP. São ainda abordadas especificamente algumas dessas estratégias meta-heurísticas. No quarto capítulo são descritas em detalhes as meta-heurísticas implementadas, e no quinto são analisados os resultados computacionais obtidos. Em seguida são apresentadas as conclusões a que se chegou. Finalmente são citadas as referências bibliográficas.

2. PROBLEMA DE DESIGNAÇÃO QUADRÁTICA – QAP

2.1. Introdução

O Problema de Designação Quadrática (ou QAP, de *Quadratic Assignment Problem*) consiste em encontrar o posicionamento ótimo em casos de designação de instalações a localidades pré-definidas, levando-se em conta as distâncias entre as localidades e os níveis de interação entre as instalações. Os termos instalação e localidade devem ser entendidos num grau abstrato e não no sentido normalmente usado, devido à grande quantidade de elementos da realidade que podem assumir essas denominações.

Vários problemas reais e de importância atual podem ser modelados como um QAP. Dentre estes, problemas de localização de fábricas ou máquinas, projetos arquiteturais (Elshafei, 1977, Gero *et al.*, 1997) e projetos de eletrônica (Junger *et al.*, 1994). Além disso, problemas clássicos de Otimização como o Problema do Caixeiro Viajante e o Problema de Partição de Grafos podem ser vistos como casos específicos do QAP (Dell'Amico *et al.*, 1997).

Uma motivação adicional para a pesquisa e o desenvolvimento de técnicas para a resolução do QAP é o fato de que este problema faz parte da classe NP-difícil (Garey and Johnson, 1979). Tal classe é composta de problemas para os quais é improvável a existência de um algoritmo eficiente que os resolva exatamente.

A seguir, são apresentadas algumas formulações do QAP.

2.2. Modelagens do QAP

2.2.1. Formulação proposta por Koopmans e Beckmann, 1957

Koopmans e Beckmann, 1957, introduziram o QAP como a designação de um determinado conjunto de instalações (fábricas, por exemplo), entre as quais são conhecidos os fluxos, a um determinado conjunto de localidades, entre as quais são conhecidas as distâncias. O objetivo é então minimizar um somatório que vincula os fluxos às distâncias. A formulação proposta por Koopmans e Beckmann, 1957, pode ser descrita como a seguir:

Sejam:

n - número de instalações ou localidades;

$V(n)$ - o conjunto de todas as permutações de $\{1, 2, \dots, n\}$;

$A(a_{ij})$ - uma matriz $n \times n$, onde a_{ij} representa o fluxo de materiais da instalação i para a j ;

$B(b_{kl})$ - uma matriz $n \times n$, onde b_{kl} representa a distância entre as localidades k e l .

O objetivo é minimizar a função

$$C(v) = \sum_{i=1}^n \sum_{j=1}^n a_{ij} b_{v(i)v(j)} \quad v \in V(n) \quad (1)$$

O objetivo é posicionar n instalações que produzam diferentes produtos, levando-se em conta que os produtos originados em uma dessas instalações poderão ser insumos para as demais e vice-versa. Haverá então fluxos de transporte e comunicação entre as mesmas. É razoável portanto imaginar que instalações que interagem muito entre si devam ser posicionadas em localidades mais próximas umas das outras. Dessa maneira, se as instalações i e j forem posicionadas nas localidades k e l , respectivamente, é natural que o custo dessa atribuição seja o produto do fluxo entre as instalações pela distância entre as localidades, desde que se considerem os custos de transporte proporcionais às distâncias percorridas. Assim, minimizar a função objetivo dada pelo somatório do produto dos fluxos pelas distâncias (1) é determinar uma permutação de instalações sobre localidades, de tal forma que esse somatório seja o menor possível. O processo de minimização deve levar em conta que as distâncias entre as localidades e os fluxos entre as instalações são dados, e portanto a estratégia a ser seguida deve posicionar os pares de instalações com maior fluxo nos pares de localidades que distem menos entre si - observando o comportamento da função objetivo - pelo uso das permutações. Em outras palavras, nessa formulação a decisão está presente na fase de permutações, escolhendo-se no conjunto V aquela que fará o sistema resultar num posicionamento ótimo.

2.2.2. Uma Formulação por Programação Inteira

Uma maneira de modelar-se o QAP é como um problema de Programação inteira, descrito a seguir (Nemhauser e Wolsey, 1988).

Sejam:

m - quantidade de instalações;

n - quantidade de localidades (com $m \leq n$);

C_{ij} - o volume transportado da instalação i para a instalação j , ou seja, o fluxo de insumos provenientes da instalação i para a instalação j ;

D_{kl} - o custo de transporte de insumos da localidade k para a localidade l , ou seja, a distância entre essas duas localidades, levando-se em conta que os custos de transporte sejam proporcionais às distâncias percorridas.

Seja a seguinte variável de decisão:

$X_{ik} = 1$ se a instalação i estiver posicionada na localidade k ;
0 caso contrário.

Temos assim a seguinte formulação:

$$\min \sum_{\substack{i,j=1 \\ i \neq j}}^n \sum_{\substack{k,l=1 \\ k \neq l}}^m C_{ij} \cdot D_{kl} \cdot X_{ik} \cdot X_{jl} \quad (2)$$

Sujeito a:

$$\sum_{i=1}^m X_{ik} \leq 1 \quad k = 1, \dots, n \quad (3)$$

$$\sum_{k=1}^n X_{ik} = 1 \quad i = 1, \dots, m \quad (4)$$

$$X_{ik} \in \{0,1\} \quad (5)$$

Na formulação acima, o conjunto de restrições (3) garante que cada localidade é designada a no máximo uma instalação, e (4) garante que toda instalação é designada a uma localidade. Note que nessa formulação o número de instalações e localidades pode ser diferente ($m \leq n$). Em geral, na literatura sobre QAP considera-se que $m=n$.

2.3. Uma Ilustração do QAP

A seguir é apresentado um exemplo simples do QAP. São 5 localidades, 3 instalações e 3 fluxos de insumos entre as mesmas (Figura 2.1):

Figura 2.1. Um exemplo do QAP

Uma solução factível para o exemplo da Figura 2.1. pode ser vista na Figura 2.2.

Figura 2.2. Uma solução factível para o exemplo mostrado na Figura 2.1

Note que o fato das instalações A, B e C terem sido posicionadas nas localidades 1, 2 e 3, respectivamente, é apenas uma coincidência. Uma outra solução factível seria aquela onde manteríamos as instalações A e C nas mesmas localidades, e posicionaríamos a instalação B na localidade 5, como representado na Figura 2.3.

Figura 2.3. Outra solução factível para o exemplo mostrado na Figura 2.1.

A solução mostrada na Figura 2.3. terá um valor maior para a função objetivo, já que a localidade 5 apresenta custos maiores do que aqueles referentes à localidade 2, em relação às outras duas. Essa segunda solução seria então pior que a primeira.

2.4. Resolução do QAP

Como dito anteriormente, o QAP pertence à classe NP-difícil. Para que se tenha uma idéia, algoritmos exatos conseguem solucionar exemplos do QAP com desempenho razoável, apenas quando esses exemplos apresentam matrizes de fluxos e distâncias de ordem máxima igual a 25 segundo Burkard e Çela, 1997, ou ainda 30, segundo Merz e Freisleben, 1999.

Por outro lado, na área de pesquisa de métodos heurísticos para problemas de Otimização, têm crescido o desenvolvimento e a aplicação das chamadas meta-heurísticas. Meta-heurísticas (Reeves, 1993 e Diaz *et al.*, 1996) são métodos que têm como objetivo guiar procedimentos de busca heurísticos, tentando explorar de modo mais eficiente o espaço de soluções.

Devido à complexidade de resolução do QAP, os trabalhos que abordam sua resolução envolvem métodos heurísticos incorporando meta-heurísticas. Alguns exemplos de meta-heurísticas utilizadas para resolução do QAP são: “Simulated Annealing” (Wilhelm e Ward, 1987), Busca Tabu (Shorin-Kapov, 1994, Taillard, 1991), Grasp (Li, Pardalos e Resende, 1994), Algoritmos Genéticos (Tate e Smith, 1995) e Algoritmos Meméticos (Merz e Freisleben, 1997, 1999, 2000). Outras referências a métodos que incorporam meta-heurísticas podem ser encontradas em (Merz e Freisleben, 1997, 1999, 2000).

No próximo capítulo descrevemos as características mais importantes das meta-heurísticas utilizadas nesse trabalho.

3. META-HEURÍSTICAS

3.1. Introdução

Nesse capítulo são definidos problemas de Otimização, Ótimos Locais e Globais, Algoritmos Heurísticos e Meta-heurísticas. Além disso, são descritas em maiores detalhes as meta-heurísticas utilizadas nesse trabalho.

3.2. Problemas de Otimização

Segundo Beasley (Reeves, 1993), muitos pesquisadores estudaram nas últimas décadas a questão de se encontrar soluções ótimas para problemas que pudessem ser modelados por uma função de variáveis de decisão e, em grande parte dos casos, fossem sujeitos a restrições. Esses problemas são chamados problemas de Otimização e geralmente são modelados como a seguir:

Minimizar $f(x)$

Sujeito a: $g_i(x) \geq b_i$ $i = 1, \dots, m;$

$h_j(x) = c_j$ $j = 1, \dots, n$

Nessa formulação geral, x é uma variável de decisão e $f(\cdot)$, $g_i(\cdot)$ e $h_j(\cdot)$ são funções gerais. Ainda, deve-se notar que a formulação supõe uma minimização da função e que para a configuração de maximização as alterações são naturais.

Há uma série de classes específicas de problemas de Otimização e o que as define são as restrições (lineares ou não), a função objetivo considerada (linear ou não) e os valores que as variáveis de decisão (contínuas ou discretas) podem assumir.

Uma classe importante dentre os problemas de Otimização é formada pelos problemas de Otimização Combinatória, caracterizados por apresentar um número finito porém imensamente grande de soluções factíveis.

Definidos os problemas de Otimização Combinatória, é importante definir os conceitos de Ótimos Locais e Globais.

3.3. Ótimos Locais e Globais

Uma característica comum em problemas de Otimização é a existência de soluções factíveis chamadas ótimos locais. Como o objetivo final de um método de resolução desse tipo de problema é a obtenção de uma solução ótima global, a existência de ótimos locais deve ser considerada e algoritmos que não os evitam podem ser tidos como ruins ou incompletos. Algoritmos eficazes devem apresentar maneiras de se desvencilhar dos ótimos locais e procurar soluções boas em novas regiões do espaço de soluções, onde possa haver soluções factíveis que melhorem o valor da função objetivo. Um conceito importante neste contexto é o de Vizinhaça.

De acordo com a literatura pesquisada, uma vizinhaça $N(x, A)$ de uma solução x é um conjunto de soluções que pode ser alcançado a partir de x , através de uma operação simples A . Essa operação A pode ser tanto a inclusão, quanto a remoção de um “objeto” da solução x , por exemplo. Outra forma de alterar uma solução x é a troca de dois “objetos” já pertencentes à mesma. Essas operações são normalmente chamadas de Movimentos, principalmente numa das meta-heurísticas a ser tratada à frente, denominada Busca Tabu. Ainda, se uma solução factível y é melhor que qualquer outra na vizinhaça $N(y, A)$, então y é um ótimo local em relação a essa vizinhaça.

3.4. Heurísticas

O termo Heurística vem do grego *heuriskein* e significa originalmente encontrar ou descobrir. É muito importante diferenciar o significado de Heurística em estudos relacionados à Inteligência Artificial daquele que deve ser dado nesse texto sobre Pesquisa Operacional. Em Inteligência Artificial, uma heurística é notadamente um método usado para que se encontre um ótimo global, como por exemplo um método *branch-and-bound*. Segundo Beasley (Reeves, 1993), essa definição é bastante razoável, mas em Pesquisa Operacional, quando se trata de uma heurística, está sendo citado um método de busca, o qual não garante que se encontre resultado algum.

Dessa forma, no contexto comum de Otimização, o termo heurística é usado como contraste a métodos que garantam uma solução global ótima.

Ainda segundo Beasley (Reeves, 1993), uma heurística é uma técnica que busca soluções boas (próximas à ótima) a um custo computacional razoável, não sendo capaz de garantir factibilidade ou otimalidade, ou mesmo, em muitos casos, de localizar o quanto próxima uma solução factível particular está do ótimo global.

Uma estratégia possível para solucionar um determinado problema de Otimização é listar todas as soluções factíveis do mesmo, avaliar as respectivas imagens da função objetivo e tomar a melhor. Todavia, é fácil perceber que essa estratégia, denominada enumeração completa ou explícita, é ineficaz. Apesar de a princípio qualquer problema de Otimização poder ser resolvido assim, na prática isso não é verdadeiro, pois qualquer problema de tamanho razoável apresenta uma quantidade enorme de soluções factíveis.

Diante disso e da complexidade de certos problemas, para os quais o uso de algoritmos exatos é ineficiente, há a possibilidade de se usar algoritmos heurísticos, capazes de encontrar boas soluções e até mesmo ótimas, inclusive para os problemas de complexidade alta.

Segundo Diaz *et al.*, 1996, o uso de algoritmos heurísticos como busca da solução de problemas de Otimização pode e deve ocorrer num número considerável de situações, o que pode ser visto nos casos a seguir:

Caso 1: Quando não há ou não é conhecido um método exato de solução, ou este requer um gasto muito grande de tempo e ou memória. Nessa situação é preferível a obtenção de uma solução considerada boa, de alguma maneira, do que não se ter solução alguma.

Caso 2: Quando para a aplicação em questão não se fizer necessária a otimalidade da solução. Esta é a situação na qual os valores obtidos para a função objetivo são relativamente pequenos, sempre. Então, supõe-se que a busca pela solução ótima do problema, que acarretará custos de tempo e memória extras, não seja uma boa alternativa se comparada ao resultado similar de uma busca mais simples que gere uma solução suficientemente boa. Em muitas aplicações, a simples melhora da solução existente é suficiente para os requisitos propostos.

Caso 3: Quando os dados que se possui não são confiáveis ou representativos, ou quando o modelo usado é uma simplificação da realidade, buscar uma solução ótima não é uma boa estratégia, pois para isso seria necessário um processamento maior e, devido à situação, a solução ótima seria tão **aproximada** quanto uma solução boa qualquer.

Caso 4: Quando há limitações de tempo, espaço para armazenamento dos dados e outras. Essas limitações exigem então o uso de técnicas eficientes ou que dispensem uma maior precisão.

Caso 5: Como passo interno a outro algoritmo. Às vezes parte-se de soluções heurísticas empregando-se métodos exatos para a continuação. Naturalmente, estando a solução inicial próxima ao valor ótimo, o algoritmo exato poderá funcionar bem em muitas situações.

3.4.1. As classes de problemas P e NP

Podemos dizer que um problema pertence à classe P (polinomial), se um algoritmo para sua resolução necessita de um tempo polinomial de execução $O(p(n))$, onde $p(n)$ é uma função polinomial e n é uma medida para o tamanho do *input* do problema. Por outro lado, se diz pertencer à classe NP (não determinístico polinomial), se existe um algoritmo de tempo polinomial que confirme se determinada resposta para o problema é uma resposta válida ou não. Um dos problemas que pertencem à classe NP é o Problema do Caixeiro Viajante. Tal problema pode ser definido como: dados um conjunto finito de pontos (cidades) V e os custos (distâncias) C_{ij} entre cada par de pontos i, j pertencente a V , determinar o menor caminho que passe exatamente uma vez por cada ponto de V .

Dentre os problemas que são ditos pertencentes à classe NP, destaca-se um grupo chamado NP-Completo. Nesse caso é equivalente dizer que se for encontrado um algoritmo de resolução em tempo polinomial para qualquer problema pertencente a essa classe, então existirá um algoritmo polinomial para a resolução de todos os problemas pertencentes à mesma. Exemplos de problema NP-Completo são o Problema do Caixeiro Viajante e o próprio QAP, um dos objetos dessa dissertação.

Desde que muitos problemas clássicos, tais como o do Caixeiro Viajante e os de Programação Inteira em geral, pertencem a essa classe (NP-Completo), encontrar um algoritmo de tempo polinomial é muito improvável, e isso implica que na resolução desses problemas deve-se investir no desenvolvimento de algoritmos de aproximação e meta-heurísticas.

3.4.2. Tipos de Heurísticas

São cinco os tipos de algoritmos heurísticos segundo Diaz *et al.*, 1996, e, ainda segundo os autores, há sempre a possibilidade de mesclá-los. Deve-se dizer também que a classificação é quanto à maneira que buscam soluções e as constróem:

Tipo 1 - Métodos construtivos: são os algoritmos heurísticos que, a cada passo, acrescentam um novo item à solução atual e verificam a factibilidade da mesma. O chamado Algoritmo Guloso é o exemplo mais famoso do tipo 1, e a lógica deste algoritmo consiste em buscar o maior benefício a cada movimento.

Tipo 2 - Métodos de decomposição: aqui, divide-se o problema em subproblemas fazendo-se com que a saída de um seja a entrada do seguinte. Ao final, tendo sido solucionados todos os problemas menores, tem-se a solução do problema inicial (que engloba os menores). Determinados autores (Ball e Magazine, 1981) separam as definições de métodos de decomposição (onde os subproblemas são resolvidos em série) e métodos de partição (onde os subproblemas são tratados paralelamente).

Tipo 3 - Métodos de redução: este tipo de algoritmo procura detectar característica(s) que a solução ótima certamente assumirá. Isso pode implicar, por exemplo, que uma variável de um problema de programação inteira sempre apresente o valor zero, ou que se associe o comportamento de uma variável ao de outra(s), criando entre essas variáveis uma dependência.

Tipo 4 - Métodos que manipulam o modelo original: aqui, transforma-se o modelo original a ser solucionado e busca-se a solução do novo problema. Ao encontrá-la, tenta-se transformá-la inversamente e torná-la a solução realmente procurada desde o início.

Tipo 5 - Métodos de busca em vizinhança: este é seguramente o tipo que apresenta melhores resultados, em geral, e portanto o de maior apelo entre os pesquisadores. Os algoritmos desse tipo caracterizam-se por partir de uma solução factível inicial (que pode ser obtida através de uma outra heurística) e alterá-la iterativamente até a ocorrência de um determinado fato, definido como critério de parada do algoritmo.

3.5. Meta-heurísticas

Existe a possibilidade de que um algoritmo heurístico seja guiado por outro. Essa é uma das situações em que ocorre a já citada mescla de estratégias heurísticas. Por exemplo, pode-se imaginar um algoritmo do Tipo 5, que, a cada iteração, procure primeiro melhorar ostensivamente a solução atual, levando em conta apenas a vizinhança da mesma e, depois de ocorrida essa melhora entre num outro processo, mais elaborado, que permita o percurso por outras regiões do espaço de soluções, e assim sucessivamente.

Essa segunda parte da iteração funcionaria como um guia para o processo de melhora ostensiva, pois normalmente apresenta passos refinados e amplos que agilizam o andamento da busca. Seria como um leme (segunda parte da iteração) orientando um motor (primeira parte).

Além disso, a segunda parte do algoritmo imaginado, por ser mais criativa em relação ao espaço de soluções e por guiar o processo de melhora local, normalmente recebe o nome de Meta-heurística. O prefixo *meta* indica superioridade ou amplidão, e é usado para determinar que esse tipo de algoritmo heurístico cumpre um papel de supervisão no processo de busca.

A seguir são descritas as meta-heurísticas utilizadas nesse trabalho, a saber, Busca Tabu, Algoritmos Genéticos e Algoritmos Meméticos.

3.5.1. Busca Tabu

A estratégia de Busca Tabu originou-se de conceitos segundo os quais métodos heurísticos conseguiriam boas soluções através da busca em regiões pouco visitadas ou “esquecidas”, da ultrapassagem de limites de factibilidade ou otimalidade local, normalmente considerados limites de Otimização. Buscas e ultrapassagens definidas por restrições que permitem o acesso a essas regiões do espaço de soluções, às quais normalmente não se percorre em outros métodos devido às características de repetição e ou diminuição de passos e visitas que tais métodos costumam apresentar no decorrer das iterações.

A maneira com que se define o algoritmo de Busca Tabu atualmente é devida a Glover, 1997, e a Hansen, 1986. Foram ainda somadas contribuições ao passar dos anos, o que permitiu a essa estratégia bons resultados em problemas de Otimização.

A técnica de Busca Tabu é, antes de tudo, baseada no conceito de histórico ou “memória” do algoritmo implementado. Essa proposta está relacionada ao conceito de memória e às estruturas com que será simulada essa característica. Tais estruturas de memória internas à técnica de Busca Tabu referem-se a quatro aspectos das soluções com as quais o algoritmo venha interagir: o quanto a solução é recente no histórico de execução (a memória do algoritmo propriamente dita, que precisa estar disponível em forma de estrutura de dados); a frequência dessa solução; sua qualidade e seu impacto nos testes de otimalidade. A partir dessas decisões o algoritmo então define quais soluções são tabus (o que significa não serem consideradas por um tempo determinado) e quais são aceitáveis (principalmente as pouco ou não visitadas ainda). Não se deve confundir uma solução tabu com uma inactivável.

Em Busca Tabu não se considera a idéia de evitar ótimos locais usando funções de probabilidade, sendo que a grande maioria dos algoritmos que seguem essa visão de Glover, 1997, é em grande parte ou totalmente determinística. Ao invés de seguirem algum acaso na aceitação de soluções piores, implementações de Busca Tabu utilizam-se maciçamente das estruturas de memória acima citadas, evitando visitas repetitivas a uma pequena região do espaço de soluções, buscando ultrapassar fronteiras e evitar otimalidade local, seguindo as indicações do histórico sobre vizinhanças promissoras e, deve-se enfatizar novamente, pouco visitadas. Como sempre há exceções, no ramo de Busca Tabu existem algoritmos classificados como da ordem de Buscas Tabu Probabilísticas.

No uso da memória e do histórico no processamento, algoritmos que seguem o conceito de Busca Tabu modificam a noção de vizinhança e substituem o termo $N(\text{solução_atual})$ por outro que também denota a vizinhança, mas que leva em conta o histórico: $N(H, \text{solução_atual})$. Assim, a estrutura histórico (H) determina as soluções factíveis que podem ser atingidas por um movimento a partir da solução atual. No algoritmo, isso significa seleccionar a “próxima_solução” da vizinhança $N(H, \text{solução_atual})$. Deve ficar claro que essa nova formulação de vizinhança é um subconjunto da anterior, e que através dos recursos de memória inerentes à Busca Tabu é que se define quais soluções factíveis vão sendo excluídas ou incluídas no novo arranjo com histórico. É o histórico que define quais soluções são atualmente tabus e quais não são. É o ponto mais decisivo da estratégia de Busca Tabu em termos de implementação ou de estruturas de dados.

- **Memória de curto prazo**

Na teoria sobre Busca Tabu é muito importante diferenciar os termos memória de curto prazo e memória de longo prazo. Cada uma dessas classificações refere-se a uma forma de se considerar a estrutura de memória dos algoritmos de Busca Tabu.

A chamada memória de curto prazo está sempre relacionada aos atributos de soluções modificados recentemente. Esses atributos cuja modificação é considerada próxima segundo um certo critério, são denominados tabu-ativos e as soluções que os contêm (ou contêm parte da estrutura dos mesmos) são então classificadas como tabus. Essas denominações e classificações ocorrem em nível de implementação normalmente, através de atributos especiais no caso das soluções-tabu e sub-atributos no caso daqueles atributos tabu-ativos. Dessa forma, é possível “eliminar” por um tempo determinado essas soluções-tabu do conjunto vizinhança $N(\text{solução})$, o que representa o foco principal da estratégia de Busca Tabu. Costuma-se atribuir ainda, certas penalidades temporárias de otimização entre as soluções-tabu, resultando assim numa gradação entre essas soluções que não serão visitadas.

- **Memória de longo prazo**

Para muitas aplicações, o uso de Busca Tabu com memória de curto prazo mostra-se efetivo para a obtenção de soluções comprovadamente boas. Ainda assim, pode-se melhorar bastante o rendimento da meta-heurística utilizando estruturas de memória de longo prazo, que não se limita a soluções visitadas tão recentemente. Quando se trata de memória de longo prazo, ainda está se fazendo referência a um novo conjunto de estratégias associadas.

Uma dessas estratégias é a de estabelecer uma memória auxiliar especializada na frequência de determinados atributos em soluções, em períodos de tempo definidos. Dessa forma o algoritmo consegue dividir o espaço de soluções em regiões que apresentam um comportamento semelhante e assim refinar o processo de determinação de soluções-tabu, agora com informações mais amplas sobre as características do problema. A frequência dos atributos pode ser verificada em termos de transição (quando o atributo muda em uma solução) e de residência (quando um atributo permanece inalterado). Esses registros que vão sendo consultados pelo algoritmo vão determinando quais os próximos passos da busca.

Dentro dos conceitos relacionados a memória de longo prazo estão também as estratégias de intensificação e de diversificação.

As estratégias de intensificação consistem no estabelecimento de mecanismos eletivos das próximas soluções a ser visitadas, de uma forma que o algoritmo passe a ser aplicado dentro das regiões que se apresentaram melhor em relação ao objetivo de otimização.

As de diversificação, ao contrário, foram projetadas para a busca em regiões novas, o que significa geralmente a inclusão de atributos pouco comuns durante a execução.

Deve ficar claro que nos dois casos o conceito inicial de Busca Tabu não é perdido, ou seja, tanto num tipo de estratégia quanto no outro se escolhe conjuntos de soluções para serem considerados tabu no decorrer do processamento. A diferença entre uma estratégia e outra está no ponto de partida: uma região muito visitada (busca tabu numa região “tabu”) ou uma região nova (busca tabu simples, na verdade).

- **Critério de aspiração**

Em certo momento de uma Busca Tabu, é valioso que movimentos considerados tabu sejam analisados segundo algum critério interessante em relação ao problema a ser resolvido e, caso esse critério seja atendido, a condição de tabu desses movimentos aspire. O critério de aspiração mais natural é permitir que um movimento tabu seja realizado antes que a restrição determinada acabe, caso esse movimento resulte numa solução melhor que a melhor encontrada até então.

- **Um Pseudo-Código de Busca Tabu**

```
Si ← solucao inicial
enquanto (critério_de_parada_não_atingido) faça
    Sc ← N(H,Si) /* Pode ser incluído um critério de aspiração */
    Atualize o histórico H
    Si ← Sc
    Se f(Sc) < f(Sbest)
        Sbest ← Sc
    fim_se
fim_enquanto
```

Busca Tabu pode ser ainda aplicada em conjunto com outros métodos, como por exemplo dentro dos Algoritmos Genéticos que serão apresentados a seguir.

3.5.2. Algoritmos Genéticos

De certa forma, afirmar que o estudo sobre Algoritmos Genéticos seja exclusivamente contemporâneo não seria inteiramente verdade, pois a origem dessa pesquisa data de quase quarenta anos. Esse início, entretanto, deu-se no ramo de Inteligência Artificial, sendo que experiências de maior vulto na área de Otimização e Pesquisa Operacional só surgiram ultimamente.

Um Algoritmo Genético, no contexto de Otimização e Pesquisa Operacional, pode ser descrito como uma estratégia de direcionamento aplicada numa busca aleatória. A principal introdução a Algoritmos Genéticos foi realizada por Holland na Universidade de Michigan nos anos sessenta e setenta, e o primeiro tratamento sistemático e fortemente teórico dado a essa técnica foi publicado em seu livro *Adaptation in Natural and Artificial Systems*, publicado em 1975 (Holland, 1975). Também Goldberg, 1989, contribuiu com interessantes adendos, notadamente de preocupação mais prática, nos anos que se seguiram.

A denominação Algoritmo Genético deriva da analogia entre a representação de uma estrutura complexa através de vetores de dados, com a noção proveniente da Biologia sobre a estrutura genética de um cromossomo. No processo de evolução das espécies, por exemplo de plantas e animais, a descendência manifesta características herdadas, características essas determinadas em nível genético devido à combinação do material presente nos cromossomos dos pais. Pode-se assim observar como um processo parecido a busca por soluções boas para problemas difíceis, através da combinação de soluções já conhecidas. A analogia certamente não é perfeita, uma vez que nem mesmo o processo real de mistura cromossômica é inteiramente descrito e coberto em todos os detalhes pelos especialistas, mas foi suficientemente atrativa para que muitas pesquisas em Otimização seguissem nesse rumo.

Na maioria das aplicações, os vetores de dados ou cromossomos são estruturas que se compõem simplesmente de zeros e uns formando *strings*.

No uso de Algoritmos Genéticos, ao contrário das outras meta-heurísticas apresentadas anteriormente, uma série de soluções é utilizada a cada iteração, a qual denomina-se População, de onde (ou dentre os seus descendentes, o que é mais provável) será escolhida a melhor solução.

Segundo Reeves, 1993, uma outra característica importante dos Algoritmos Genéticos é a de não necessariamente manipularem informações específicas a respeito do problema

tratado, no decorrer do processo evolutivo dos cromossomos. Em outras palavras, a partir da representação de uma solução em um cromossomo, a melhoria dessa solução é um problema diferente e independente do problema original.

Na teoria sobre Algoritmos Genéticos, Operadores Genéticos são processos caracterizados por realizarem a manipulação da População atual, visando alguma alteração ou avaliação da mesma. Alguns dos Operadores Genéticos clássicos para a manipulação de uma população de cromossomos são: Operador de Cruzamento (que realiza o cruzamento das soluções - ou cromossomos) e Operador de Mutação (um processo aleatório de alteração de uma solução - ou cromossomo).

Continuando a analogia com a Genética, em Algoritmos Genéticos, variáveis são relacionadas a genes, os possíveis valores assumidos por uma variável relacionados a alelos e a posição de uma variável numa *string* é chamada locus (em situações simples, o locus de uma variável ou gene não é importante, ainda que em casos mais complexos seja). Ainda, em Genética a estrutura real dos cromossomos é denominada Genótipo e a aparência física baseada na mesma é dita Fenótipo. Em Otimização, quando citados no ramo de Algoritmos Genéticos, o Genótipo é a codificação das estruturas que é processada através do algoritmo, enquanto o Fenótipo é o conjunto de parâmetros compilado ou interpretado.

Enfim, quando da avaliação de uma população de cromossomos, a qualidade dos mesmos dentro do contexto de Pesquisa Operacional é dita *fitness* ou função de aptidão.

Na busca pela solução ótima de problemas de Otimização, um Algoritmo Genético, de forma geral, funciona da seguinte maneira: inicia-se com uma certa população P de cromossomos (sempre representando soluções) dos quais a função de aptidão é conhecida. A função de aptidão de cada cromossomo é, normalmente, o valor gerado pela função objetivo para a respectiva solução avaliada, ainda que outras formas de avaliação existam e possam ser empregadas com sucesso, às vezes até maior. Então é feita a escolha de dois ou mais pais segundo alguma regra, e é aplicado a esses um operador de cruzamento (mais à frente são descritos alguns tipos de operador de cruzamento).

A partir da execução do operador de cruzamento, os descendentes são incluídos na população. Nesse momento pode-se também escolher cromossomos para serem retirados da população segundo algum critério (normalmente relacionado à função de aptidão). Na sequência, o processo é repetido a cada geração (ou iteração), sendo que as regras para escolha de pais podem ser alteradas, assim como pode ser alterado o esquema de eliminação das

soluções a sair da População. Dessa forma, uma população deixa de existir totalmente ou em parte (isso dependendo do número de cruzamentos de cromossomos comparado com o tamanho da população), e uma nova população passa a ser tratada. A tendência é que esta nova população seja melhor que as anteriores, pois se trata de um modelo evolucionário baseado na idéia de sobrevivência do indivíduo mais forte. Ainda, também com alguma probabilidade de ocorrência, pode ser aplicado um operador de mutação em membros da população, visando a uma maior diversidade do conjunto. Um exemplo de operador de mutação pode ser a alteração de alelos do cromossomo segundo uma certa probabilidade, como mostra a Figura 3.1.

Cromossomo1	0 <u>0</u> 0 0 1 0 1
Cromossomo1 mutante	0 <u>1</u> 0 0 1 0 1

Figura 3.1. Exemplo de um operador de mutação

- **Alguns Tipos de Operador de Cruzamento**

Operador de Cruzamento em um ponto

Nesse tipo de operador de cruzamento, cada pai tem seus genes separados em um único ponto e então uma das metades de cada é trocada pela respectiva metade do outro. Sejam os pais P1 e P2, segue a geração dos descendentes D1 e D2 como mostra a Figura 3.2.

P1	1 0 1 0 0 1 0	D1	1 0 1 <u>1 0 0 1</u>
P2	<u>0 1 1 1 0 0 1</u>	D2	<u>0 1 1</u> 0 0 1 0

Figura 3.2. Exemplo do Operador de cruzamento em um ponto

Operador de Cruzamento em vários pontos

Semelhante ao anterior, mas com um aumento no número de divisões dos cromossomos escolhidos para cruzamento, separando-os por mais de um ponto, como mostra a Figura 3.3. Uma recombinação desse tipo traz à população uma maior quantidade de candidatos a novos membros.

P1	1 0 1 0 0 1 0	D1	1 0 1 <u>1 0</u> 1 0
P2	<u>0 1 1 1 0 0 1</u>	D2	<u>0 1 1</u> 0 0 <u>1 1</u>

Figura 3.3. Operador de cruzamento em vários pontos

Operador de Cruzamento uniforme

Esse operador não utiliza pontos de cruzamento, mas determina, através de um parâmetro global, qual a probabilidade de cada alelo ser herdado de qual pai.

Operadores de Cruzamento Específicos ao QAP

Devido à sua complexidade, determinados problemas geram trabalhos que buscam desenvolver operadores de cruzamento específicos. Para o Problema do Caixeiro Viajante, por exemplo, há uma série de operadores de cruzamento específicos tais como: Cruzamento baseado em ordem, Cruzamento baseado em posição, Cruzamento preservativo máximo e Cruzamento de alteração de margem, entre outros, apresentados em Larrañaga et al., 1999.

Isso também ocorre na pesquisa sobre o QAP. A seguir são descritos dois operadores de cruzamento específicos para o QAP.

O operador de cruzamento DPX (*distance preserving crossover*), proposto para o QAP por Merz e Freisleben, 1999, é baseado no conceito de distância entre as soluções, ou seja, no número de alelos diferentes entre duas soluções. Este tipo de cruzamento mostrou-se eficaz também para o Problema do Caixeiro Viajante, como mostrado em Freisleben e Merz, 1996a, e Freisleben e Merz, 1996b. Inicialmente, todos os posicionamentos coincidentes entre os dois pais são repetidos no descendente. Então as localidades restantes no descendente são preenchidas aleatoriamente, porém desde que seja mantida entre o descendente e cada pai a mesma distância que ocorrer entre os dois pais.

Outro operador é o chamado CX, assim chamado por ter sido proposto para a resolução do Problema do Caixeiro Viajante com o nome *Cycle Crossover* (Oliver et al., 1987). Esse operador se mostrou mais eficaz para o QAP, quando comparado ao DPX, segundo Freisleben e Merz, 2000. Dessa forma foi o operador escolhido para ser implementado nesse trabalho. No capítulo onde são descritas as meta-heurísticas implementadas, esse operador é descrito detalhadamente.

- **Um Pseudo-Código de um Algoritmo Genético**

```
Inicializa_População( )
enquanto (critério_de_parada_não_attingido)
    para cont = 1 até número_de_cruzamentos faça
        Seleciona_Pais( população )
        descendente = Cruzamento( pai1, pai2 )
        Acrescenta( descendente, população )
    fim_faça
    Mutação( população )
    população = Seleciona( população )
    se Convergiu( população ) então
        Diversifica( população )
    fim_se
fim_enquanto
```

De acordo com Diaz *et al.*, 1996, os Algoritmos Genéticos em aplicação pura têm apresentado resultados animadores, inclusive para problemas complexos em áreas de estudo como Pesquisa Operacional, Planejamento Paralelo, Eletrônica, Biologia Molecular e Físico-química, Engenharia Civil, Bancos de Dados, Geofísica e Inteligência Artificial, devido ao constante acesso do método a um grande subconjunto do espaço total de soluções, a cada iteração.

Essa possibilidade de processar mecanismos de otimização em várias soluções ao invés de uma, é a grande diferença entre os Algoritmos Genéticos e as demais meta-heurísticas não baseadas em evolução. Verifica-se então a partir disso que interações entre heurísticas podem vir a calhar também junto a esse tipo de busca. É aí que aparece a noção de Algoritmo Memético, na qual as definições básicas sobre os genéticos permanecem, porém com a adição de buscas locais ostensivas a cada iteração.

3.5.3. Algoritmos Meméticos

Os *Algoritmos Meméticos (AMs)*, denominação proposta por Moscato em 1989, são métodos meta-heurísticos de busca baseados em populações. Nesse aspecto, eles têm certa similaridade com os chamados Algoritmos Genéticos e as Estratégias de Evolução. Basicamente, os AMs utilizam uma população de “agentes”, que possuem informação sobre os métodos de resolução particulares ao domínio do problema considerado. Os AMs então

combinam períodos de busca individual desses agentes com algoritmos que trocam informação entre as soluções, em muitos casos utilizando operadores de cruzamento de soluções anteriores encontradas pelos agentes. Por esta razão, muitos pesquisadores os chamam de *Algoritmos Genéticos Híbridos (AGHs)*. Não obstante, existem algumas implementações de AMs que não têm nenhuma analogia com a Evolução Biológica, logo, é dito que em geral os AGHs estão incluídos nos AMs mas a recíproca não é verdadeira. Pelo fato de serem facilmente implementáveis em sistemas de computação paralelos, também são chamados de *Algoritmos Genéticos Paralelos*, mas essa denominação é ainda mais confusa, pois não se consegue diferenciá-los daqueles algoritmos que têm fortes analogias com a Evolução Biológica, e não usam informação do problema para fazer períodos de busca individual.

Os AMs têm se mostrado muito eficazes em vários problemas de otimização combinatória, como pode ser visto nas várias publicações encontradas na página da Internet a eles dedicada (MAs Home Page, 2002), ou ainda em (Moscato, 1999) e (Moscato e Cotta, 2001).

Segundo Moscato, 1989, os AMs podem ser vistos como uma estratégia onde os elementos da população, preferencialmente chamados de “agentes” e não “indivíduos”, passam não só a se reproduzir mas a competir e a cooperar entre si. No AM proposto por Moscato e Norman, 1989, para o problema do Caixeiro Viajante, foi implementado um critério de “vizinhança” entre os agentes. A competição se dá então entre os agentes consecutivos num anel e a cooperação entre os mais afastados. O processo de cooperação é definido pela cessão de características de um agente para a formação de um novo, enquanto que a competição constitui-se da eliminação de um agente por outro, normalmente em uma decisão baseada em funções de probabilidade, que por sua vez envolvem a função de aptidão aplicada aos agentes envolvidos ou envolvem diretamente a função objetivo do problema de otimização considerado.

O termo “Memético” é derivado de “*meme*”, que pode ser entendido como o termo análogo a gene, no contexto de Evolução Cultural. Em um Algoritmo Memético normalmente é levado em conta o conhecimento existente sobre o problema tratado, e assim as soluções descendentes são geradas a partir de soluções já bem desenvolvidas com estratégias “ad-hoc” para o problema, ao contrário dos métodos evolutivos tradicionais (Moscato, 1989, 1999 e Moscato e Cotta, 2001). Como consequência, cada agente pode fazer uso de soluções obtidas

por outros métodos. Este conhecimento pode ser: características da solução ótima, heurísticas (métodos de busca local assim como meta-heurísticas diferentes), algoritmos exatos truncados etc. Além disso, diferentes agentes podem fazer uso de diferentes algoritmos, e mesmo ter diferentes comportamentos que parametrizam o operador de cruzamento (Berretta e Moscato, 1999).

Segundo Diaz *et al.*, 1996, o êxito dos Algoritmos Meméticos está relacionado ao equilíbrio existente entre a rapidez da busca local e a ocorrência de diversidade no espaço de soluções utilizado, o que evita a convergência prematura desse método.

A seguir há o pseudo-código de um exemplo de algoritmo Memético que usa busca local no indivíduo (ou agente) gerado depois do operador de cruzamento.

- **Um Pseudo-Código de um Algoritmo Memético com busca local**

```
Inicializa_População( )
enquanto (critério_de_parada_não_atingido)
    para cont = 1 até número_de_cruzamentos faça
        Seleciona_Pais( população )
        descendente = Cruzamento( pai1, pai2 )
        descendente = Busca_Local( descendente )
        Acrescenta( descendente, população )
    fim_faça
    Mutação( população )
    população = Seleciona( população )
    se( Convergiu( população ) ) então
        Diversifica ( população )
    fim_se
fim_enquanto
```

4. META-HEURÍSTICAS IMPLEMENTADAS

4.1. Introdução

Neste capítulo são descritas as meta-heurísticas implementadas. Os métodos utilizados foram Busca Tabu, Algoritmo Genético e Algoritmo Memético, sendo que este último foi implementado em duas versões, uma utilizando Busca Tabu e outra utilizando uma Busca Local, ambas encontradas na literatura. Além da descrição dos parâmetros usados em cada estratégia, são apresentados os pseudocódigos e os respectivos comentários sobre estes últimos.

4.2. Busca Tabu

O algoritmo de Busca Tabu implementado foi desenvolvido por Taillard, 1991, e apresenta as características descritas a seguir.

- **Representação de uma solução**

Uma solução é representada por um vetor unidimensional de inteiros, onde cada posição representa uma localidade e cada inteiro armazenado representa uma instalação.

Na Figura 4.1., a instalação 2 está na localidade 1, a instalação 4 está na localidade 2 e assim sucessivamente.

2	4	7	1	8	9	3	5	6
1	2	3	4	5	6	7	8	9

Figura 4.1. Representação por vetor de uma solução factível para um exemplo do QAP de ordem 9.

- **Movimento**

Um movimento consiste na troca de instalações entre duas localidades. A Figura 4.2. apresenta um movimento, onde as localidades 3 e 8 têm suas respectivas instalações (7 e 5) trocadas em relação à solução representada pela Figura 4.1.

2	4	5	1	8	9	3	7	6
1	2	3	4	5	6	7	8	9

Figura 4.2. Movimento entre as localidades 3 e 8, em relação à solução representada pela Figura 4.1.

- **Atributo**

O atributo é um par (i,j) que representa duas localidades, que têm as instalações trocadas num movimento.

- **Critério tabu**

Quando ocorre uma troca de instalações entre um par de localidades i e j , ou seja, um movimento, essa mesma troca fica proibida de ser desfeita durante o respectivo tempo tabu.

- **Tempo tabu**

A cada movimento considerado tabu é atribuído um tempo tabu gerado aleatoriamente num intervalo (a,b) , onde tais valores estão relacionados ao tamanho do exemplo sendo resolvido. No capítulo de resultados computacionais estão os valores testados.

- **Critério de Aspiração**

Um movimento deixa de ser tabu antes da iteração determinada, quando sua escolha diminui o valor da função objetivo, comparando-se com a melhor solução encontrada até então.

- **Pseudocódigo da Busca Tabu**

início

$S_i \leftarrow \text{Gera_Solução_Inicial}()$

enquanto (critério_de_parada_não_atingido)

$(i,j) \leftarrow \text{Encontra_o_Melhor_Movimento}(S_i)$

 Realiza_movimento(i,j)

 Atualiza_Lista_tabu(i,j)

se $f(S_i) < f(S_{\text{melhor}})$ **então**

$S_{\text{melhor}} \leftarrow S_i$

fim_se

 {CONTINUA}

```

        Armazena_Diferença_Entre_os_Custos()
    fim_enquanto
fim

onde

Gera_Solução_Inicial()

    Gera uma solução inicial aleatoriamente.

Encontra_o_Melhor_Movimento()

    início
         $\Delta_{\text{melhorcusto}} \leftarrow \text{infinito}$ 
        para  $i = 1$  até  $n - 1$  faça
            para  $j = i + 1$  até  $n$  faça
                movimento  $\leftarrow (i, j)$ 
                se  $(\Delta_{\text{custo}}(i, j) < \Delta_{\text{melhorcusto}}$  e  $(i, j)$  é não tabu) ou  $((i, j)$  aspirou
                condição de tabu) então
                    melhor_movimento  $\leftarrow (i, j)$ 
                     $\Delta_{\text{melhorcusto}} \leftarrow \Delta_{\text{custo}}(i, j)$ 
                    se (aspirou condição de tabu) então
                        termina a função
                fim_se
            fim_se
        fim_para
    fim_para
    retorna melhor_movimento
fim

```

Essa função avalia todos os movimentos (i, j) com $i=1, \dots, n$ e $j=1, \dots, n$ e escolhe o movimento não tabu que fornece uma variação na função objetivo, a melhor possível ($\Delta_{\text{melhorcusto}}$). Contudo, se algum movimento tabu passa pelo critério de aspiração, o mesmo é escolhido de antemão.

Nessa busca tabu, ou é encontrado um posicionamento que vence o melhor resultado atual, ou é escolhido o melhor movimento possível, mesmo que este seja ruim, com o objetivo de se alcançar novas regiões do espaço de soluções. Pode ocorrer a situação raríssima em que todos os movimentos candidatos sejam tabu e nenhum atenda ao critério de aspiração, e nesse caso as iterações seguem até que um dos pares candidatos seja liberado.

4.3. Algoritmo Genético

O Algoritmo Genético implementado e descrito a seguir tem as características do algoritmo descrito em Merz e Freisleben, 1999.

- **Representação**

Um indivíduo é representado por um vetor unidimensional de inteiros, onde cada posição representa uma localidade e cada elemento representa uma instalação, como descrito em Busca Tabu. A população é representada por um conjunto desses vetores.

- **Seleção dos pais**

A seleção dos pais é sempre aleatória e resulta em dois escolhidos (característica básica de um algoritmo genético que pode ser alterada).

- **Operador de Cruzamento CX**

O operador de cruzamento CX foi primeiramente proposto por Oliver et al., 1987, para a resolução do Problema do Caixeiro Viajante, com o nome *Cycle Crossover*. A sua utilização do QAP é descrita a seguir.

Todas as instalações designadas para a mesma localidade nos dois pais são também designadas para essa localidade no descendente. Então, dentre as localidades restantes, uma é selecionada aleatoriamente e a instalação correspondente a um dos pais (selecionado também aleatoriamente) é designada para a localidade correspondente no descendente. Em seguida, a instalação não selecionada no passo anterior é designada para a localidade correspondente à localidade dessa instalação no pai já selecionado aleatoriamente. Esses passos são repetidos até que uma instalação já designada é encontrada e então se reinicia o procedimento, a partir da primeira localidade vazia encontrada no descendente.

Esse operador tem as características necessárias a um cruzamento, ou seja, garante que no descendente haja todas as n instalações também presentes nos pais, e que não aconteçam repetições das mesmas.

A Figura 4.3. ilustra a aplicação deste operador em um exemplo do QAP de tamanho 9. A ilustração foi retirada de Merz e Freisleben, 1999.

Figura 4.3. Exemplo do operador de cruzamento CX

Primeiramente, as instalações 4, 9 e 6, que estão posicionadas na mesma localidade nos dois pais, são herdadas pelo descendente exatamente nas localidades originais.

Em seguida é escolhida uma localidade vaga aleatoriamente, por exemplo, a 3. Então é escolhida também aleatoriamente a instalação da localidade 3 de um dos pais, por exemplo a instalação 7, que é herdada pelo descendente na sua localidade original (instalação/localidade sombreadas). Escolhendo-se a instalação 7 do pai A, a instalação 5 do pai B foi impedida de ocupar a localidade 3. Então a instalação 5 é herdada pelo descendente na localidade correspondente à do pai A, ou seja, a localidade 8. Da mesma forma, agora a instalação 2 ficou impedida de ocupar a localidade 8, e então é posicionada na localidade 1 do descendente, a mesma que ocupa no pai A. O passo seguinte seria posicionar a instalação 7, porém esta já foi posicionada. Então se escolhe a localidade vaga mais à esquerda, que é a 4, e nesta localidade é posicionada a instalação correspondente no pai B, a 8, ficando invertida a opção inicial pelo pai (instalação/localidade sombreadas). Seguindo os mesmos passos iniciais, a instalação 1 é herdada pelo descendente na localidade correspondente à do pai B, ou

seja, a localidade 7. O mesmo ocorre com a instalação 3 na localidade 5. Desta maneira, cada instalação fica posicionada numa localidade do descendente.

• Operador de Mutação

São realizadas trocas sucessivas entre pares de instalações, num número de vezes pré-determinado. A segunda instalação escolhida para troca a cada passo pertencerá ao par trocado no próximo. A distância entre o cromossomo original e o resultante da sequência de trocas é o número de trocas realizadas mais 1.

A Figura 4.4. ilustra a aplicação do operador de mutação num indivíduo. A ilustração foi retirada de Merz e Freisleben, 1999.

Figura 4.4. Exemplo do operador de mutação

Inicialmente, as instalações 9 e 4 são trocadas de localidade. Em seguida a 4 é trocada com a 1, e a 1 é trocada com a 3. A distância (número de alelos diferentes entre I1 e I2) é 3 (número de trocas) mais 1.

- **Função de Aptidão**

A função de aptidão de um indivíduo é o valor da função objetivo aplicada ao mesmo.

- **Atualização da População**

A cada geração, é selecionado um grupo de indivíduos de pior valor da função de aptidão que ultrapasse o tamanho inicial da população. Esse grupo é então eliminado.

- **Pseudocódigo do Algoritmo Genético**

início

 Inicializa_População()

enquanto (critério_de_parada_não_atingido)

para cont = 1 **até** número_de_cruzamentos **faça**

 Seleciona_Pais (população)

 descendente = Cruzamento (pai1, pai2)

 Acrescenta (descendente, população)

fim_para

 população = Atualiza_População (população)

se (Convergiu (população)) **então**

 índice_do_melhor = Encontra_Melhor (população)

para i = 1 **até** tamanho_população **faça**

se (i ≠ índice_do_melhor) **então**

 população[i] = Mutação (população[i])

fim_se

fim_para

fim_se

fim_enquanto

fim

onde

Inicializa_População()

 Gera aleatoriamente um número pré-determinado de indivíduos.

Seleciona_Pais(população)

 Seleciona dois indivíduos aleatoriamente.

Cruzamento(pai1, pai2)

Recombina os dois cromossomos pais selecionados, segundo o operador CX descrito anteriormente.

Acrescenta(descendente, população)

Acrescenta o descendente gerado à população.

Atualiza População (população)

A quantidade de cromossomos de pior função de aptidão, que ultrapasse o tamanho inicial da população, é eliminada.

Convergiu(população)

Inicialmente verifica se a distância (número de alelos diferentes) entre os indivíduos de toda a população é pequena. Essa margem é definida por uma constante. A seguir, verifica se o melhor custo atual é pouco melhor que o armazenado há um número estipulado de iterações. Essa outra margem de comparação e o número estipulado de iterações também são definidos por uma constante. Se ambas as condições forem verdadeiras, é considerado que a população convergiu.

Encontra_Melhor(população)

Encontra o indivíduo com melhor valor de função de aptidão.

Mutação(indivíduo)

Altera o posicionamento no indivíduo, segundo o operador descrito anteriormente. O operador de mutação é aplicado sobre toda a população, com a exceção do indivíduo de melhor função de aptidão.

4.4. Algoritmo Memético

Inicialmente implementou-se o Algoritmo Memético encontrado em Merz e Freisleben, 1999, que utiliza uma busca local. A seguir, substituímos a busca local pela Busca Tabu descrita na seção 4.2. Assim, os componentes dos algoritmos já foram descritos anteriormente. A seguir o pseudocódigo:

- **Pseudocódigo do Algoritmo Memético**

```

início
  Inicializa_População( )
  para i = 1 até tamanho_população faça
    população[i] ← Busca_Local ( população[i] )
  fim_para
  enquanto (critério_de_parada_não_atingido)
    para cont = 1 até número_de_cruzamentos faça
      Selecciona_Pais( população)
      descendente ← Cruzamento ( pai1, pai2 )
      descendente ← Busca_Local ( descendente )
      Acrescenta (descendente, população)
    fim_para
    população ← Atualiza_População ( população )
    se ( Convergiu( população ) ) então
      índice_do_melhor ← Encontra_Melhor( população )
      para i = 1 até tamanho_população faça
        se ( i ≠ índice_do_melhor ) então
          população[i] ← Mutação( população[i] )
          população[i] ← Busca_Local ( população[i] )
        fim_se
      fim_para
    fim_se
  fim_enquanto
fim

```

A Busca Local é usada na melhora da população inicial, dos descendentes gerados nos cruzamentos e dos descendentes resultantes das mutações. Foram implementadas duas versões de algoritmos meméticos. Na primeira, em Busca_Local() utilizou-se o algoritmo chamado Busca Local *Fast-2-Opt*, encontrado em Merz e Freisleben, 2000. A segunda versão de algoritmo memético utiliza a Busca Local Tabu descrita na seção 4.2. A seguir é descrita a Busca Local *Fast-2-Opt*.

A busca local *Fast-2-Opt* realiza o primeiro movimento simples (troca de duas instalações entre as respectivas localidades) que melhore o melhor custo atual. Isso significa que essa busca não avalia a qualidade dos possíveis movimentos seguintes, o que acelera seu desempenho sem grande perda na qualidade das soluções. Para que fosse acelerada ainda mais, e ainda sem perda de qualidade, usou-se a técnica “*don't look bits*”, que consiste em

desconsiderar a localidade i na iteração corrente caso esteja marcada por um *bit* proibitivo num vetor auxiliar. A localidade i é marcada no vetor auxiliar com o *bit* de proibição, se não fizer parte do movimento de melhora anterior. Quando um movimento de melhora é realizado, os *bits* auxiliares das duas localidades envolvidas são liberados.

Busca_Local_Fast-2-Opt()

início

Todas as localidades iniciam liberadas

enquanto(critério_de_parada_não_attingido)

para $i = 1$ até n **faça**

se(localidade não está proibida) **então**

para $j = 1$ até n **faça**

se(se o movimento (i,j) melhora a solução atual) **então**

 Realiza_movimento(i, j)

 Libera i e j

 A Busca Local é encerrada

senão

 Proíbe localidade i

fim_se

fim_para

fim_se

fim_para

fim_enquanto

fim

Note que se a localidade i valer 1 e for proibida, e numa iteração posterior o movimento $i = 3$ e $j = 1$ for realizado, o índice 1 do vetor auxiliar (ou seja, a localidade 1) será liberado. Em outras palavras, por haver um único vetor auxiliar, uma instalação escolhida libera a localidade de mesmo índice, se esta estiver proibida.

No capítulo seguinte estão descritos os testes e resultados computacionais obtidos.

5. RESULTADOS COMPUTACIONAIS

5.1. Introdução

Nesse capítulo são apresentados os resultados computacionais obtidos na resolução do QAP usando as implementações das meta-heurísticas pesquisadas.

Os testes foram realizados num microcomputador com processador Pentium II 400 MHz e 128 Mbytes de memória RAM, e as implementações foram realizadas em linguagem C.

As heurísticas foram testadas usando-se 17 exemplos numéricos do QAP, providos pelo *site* QAPLib (QAP Library) cujo endereço é <http://www.opt.math.tu-graz.ac.at/qaplib/>. Os exemplos numéricos são de tamanho 25 até 150, ou seja, exemplos considerados difíceis para algoritmos exatos. O tamanho de cada exemplo é indicado pelo número que consta de seu nome. Por exemplo, *lipa90a* é de tamanho 90, ou seja, apresenta 90 instalações a serem designadas a 90 localidades. Assim como *lipa90a*, todos os exemplos usados apresentam um número igual de instalações e localidades. Os exemplos numéricos foram escolhidos de acordo com a literatura pesquisada (Taillard, 1991, Merz e Freisleben, 1997, 1999, 2000) e os resultados obtidos são comparados com os melhores resultados encontrados até o momento.

As heurísticas foram executadas por um número fixo de iterações ou por um tempo computacional pré-fixado. A Tabela 5.1 descreve a nomenclatura utilizada em todas as tabelas desse capítulo.

Exemplo	Nome do exemplo numérico
Melhor_sol	Valor da f.o. para a melhor solução conhecida
Gap	Diferença percentual entre a solução da heurística e a Melhor_sol
It_Melhor	Iteração onde foi encontrada a melhor solução pela heurística
It_total	Quantidade total de iterações
T_melhor	Tempo onde foi encontrada a melhor solução pela heurística
T_total	Tempo total de execução

Tabela 5.1. Descrição das legendas utilizadas na descrição dos resultados

Note que T_{total} será indicado apenas nas tabelas onde a heurística foi executada com critério de parada sendo quantidade de iterações, e It_{total} apenas quando o critério de parada foi o tempo total de execução.

5.2. Busca Tabu

A seguir são apresentados os resultados obtidos com a implementação da Busca Tabu tratada em Taillard, 1991, para a resolução do QAP.

A Tabela 5.2 mostra os resultados da Busca Tabu ao final de 3000 iterações para cada exemplo. O Tempo Tabu foi atribuído aleatoriamente a partir de um intervalo fixado. Foram testados diferentes intervalos de valores para a atribuição do Tempo Tabu: $[0.6n, 5n]$, $[0.6n, n-1]$, $[0.6n, 2n]$, $[5n, 10n]$ e $[0.9(n-1), 1.1(n-1)+4]$. A última faixa de intervalo foi a utilizada por Taillard, 1991. A que obteve melhores resultados foi a faixa $[0.6n, 10n]$. Conseqüentemente, os resultados mostrados na Tabela 5.2 são os obtidos a partir desse intervalo.

Exemplo	Melhor_Sol	Gap	It_Melhor	T_melhor	T_total
chr25a	3796	10,06	671	0,16	0,63
bur26a	5426670	0,09	29	0,02	0,68
Nug30	6124	0,56	563	0,20	0,92
kra30a	88900	1,80	438	0,17	0,94
ste36a	9526	3,11	657	0,32	1,35
tai60a	7208572	1,92	436	0,65	4,07
tai80a	13557864	1,52	505	1,43	7,84
tai100a	21125314	1,37	1185	5,15	13,09
sko100a	152002	0,61	1128	5,11	13,21
tai60b	608215054	0,80	2942	4,43	4,60
tai80b	818415043	3,74	2917	7,87	8,22
tai100b	1185996137	6,48	1975	8,88	13,42
wil100	273038	0,37	1650	7,36	13,32
tho150	8133484	0,59	1919	22,81	35,82
tai150b	498896643	3,43	2136	25,10	35,57
lipa90a	360630	0,57	686	2,36	10,32
lipa90b	12490441	0,00	527	1,99	1,99
Média		2,18	1197	5,53	9,76

Tabela 5.2. Resultados de Busca Tabu em 3000 iterações

Uma das tentativas de variação usadas na configuração acima foi limitar a vizinhança visitada pela Busca Tabu, no momento da escolha do melhor movimento possível. Tanto a limitação da vizinhança em $n/2$ (metade do tamanho do exemplo), quanto em $0.75n$, não surtiram bons resultados, sendo que a limitação em $0.75n$ foi melhor que em $n/2$.

Na Tabela 5.3, a seguir, são mostrados os resultados da mesma Busca Tabu, porém com o tempo de processamento sendo o critério de parada e não o número de iterações. O tempo máximo fixado foi de 5n segundos.

Exemplo	Melhor_Sol	Gap	It_Melhor	It_total	T_melhor
chr25a	3796	1,84	3244	606950	0,66
bur26a	5426670	0,00	86112	86112	19,21
nug30	6124	0,00	4852	4852	1,44
kra30a	88900	0,00	23932	23932	7,12
ste36a	9526	0,00	46223	46223	20,16
tai60a	7208572	1,19	124753	224225	166,81
tai80a	13557864	0,88	149734	154471	387,66
tai100a	21125314	0,77	17411	113493	76,65
sko100a	152002	0,07	39032	114427	170,37
tai60b	608215054	0,61	104486	225166	139,07
tai80b	818415043	3,39	139859	156011	358,48
tai100b	1185996137	5,93	16243	114797	70,69
wil100	273038	0,03	47070	114354	205,58
tho150	8133484	0,13	16333	63620	192,93
tai150b	498896643	2,39	57364	63602	676,47
lipa90a	360630	0,50	83122	131377	284,73
lipa90b	12490441	0,00	527	527	1,78
Média		1,04	56488	132008	163,52

Tabela 5.3 – Resultados de Busca Tabu em 5n segundos

5.3. Algoritmos Genéticos

A seguir são apresentados os resultados obtidos com a implementação de Algoritmos Genéticos, apresentados no capítulo anterior, para a resolução do QAP.

A Tabela 5.4. apresenta os resultados do Algoritmo Genético com uma população de 40 indivíduos e uma quantidade de cruzamentos igual a 20 (tamanho da população/2). Esses são os parâmetros para os quais o algoritmo genético obteve os melhores resultados.

Exemplo	Melhor_Sol	Gap	It_Melhor	It_total	T_melhor
chr25a	3796	49,42	16763	18048	116,08
bur26a	5426670	0,26	17462	17648	128,64
nug30	6124	4,18	10323	17892	86,50
kra30a	88900	4,90	15988	17859	134,27
ste36a	9526	14,59	14520	15985	163,50
tai60a	7208572	7,08	13657	14161	289,30
tai80a	13557864	6,48	11129	11359	391,90
tai100a	21125314	7,34	9090	10452	434,98
ska100a	152002	6,39	9361	10438	448,50
tai60b	608215054	6,37	13034	13759	284,01
tai80b	818415043	10,08	11905	11931	399,14
tai100b	1185996137	11,36	9757	10257	475,64
wil100	273038	2,87	10177	10295	494,39
tho150	8133484	8,17	7573	7829	725,67
tai150b	498896643	11,19	7811	7831	748,18
lipa90a	360630	1,06	10809	11235	432,94
lipa90b	12490441	24,41	7683	11241	307,63
Média		10,36	11590	12836	356,54

Tabela 5.4. Resultados de GA em 5n segundos

A Tabela 5.5. mostra a média de resultados, alterando-se o tempo máximo de execução de 5n segundos para 50n segundos:

T_total	Gap	It_Melhor	It_total	T_melhor
5n	10,36	11590	12836	356,54
50n	7,56	87920	117048	3434,96

Tabela 5.5. Resultados médios de GA em 5n e 50n segundos

A Tabela 5.6 apresenta os resultados médios com duas diferentes estratégias de mutação. A princípio o operador de mutação é utilizado apenas quando a população converge (M1). Em M2 o operador de mutação é utilizado sem a necessidade da população ter

convergado. Nesse último caso, a cada geração foram realizadas 40 mutações em indivíduos escolhidos aleatoriamente.

Mutação	T_total	Gap	It_Melhor	It_total	T_melhor
M1	5n	10,36	11590	12836	356,54
	50n	7,56	87920	117048	3434,96
M2	5n	9,33	5511	5933	362,33
	50n	6,24	46315	52554	3563,51

Tabela 5.6. Resultados médios de GA usando diferentes estratégias para mutação

A estratégia M1 é a apresentada por Merz e Freisleben, 1999, e foi a melhor estratégia para o algoritmo memético, que possui uma busca local. Já a estratégia M2 obteve melhores resultados no algoritmo genético, indicando a necessidade de uma maior aplicação do operador de mutação quando não é utilizada uma Busca Local.

5.4. Algoritmos Meméticos

A seguir são apresentados os resultados obtidos pelos dois algoritmos meméticos. Na Tabela 5.7 estão os resultados obtidos pelo AM, utilizando-se a busca local proposta por Merz e Freisleben, 1999 (MBL). Na Tabela 5.8 estão indicados os resultados do AM utilizando-se como busca local a busca tabu de Taillard, 1991 (MBT). Em ambas, o critério de parada foi o tempo de execução igual a 5n segundos.

Exemplo	Melhor_Sol	Gap	It_Melhor	It_total	T_melhor
Chr25a	3796	0,00	43	43	3,83
Bur26a	5426670	0,08	16	4211	0,90
Nug30	6124	0,07	16	3343	2,31
Kra30a	88900	1,35	32	1104	4,52
Ste36a	9526	0,00	47	47	11,32
Tai60a	7208572	1,47	86	316	85,54
Tai80a	13557864	1,01	103	173	242,91
Tai100a	21125314	1,05	100	102	493,20
sko100a	152002	0,14	72	102	355,80
tai60b	608215054	0,02	62	861	60,19
tai80b	818415043	0,57	138	153	362,99
tai100b	1185996137	0,16	75	96	392,79
wil100	273038	0,23	55	101	278,06
tho150	8133484	0,39	40	41	744,15
tai150b	498896643	0,80	45	45	766,58
lipa90a	360630	0,47	128	130	443,56
lipa90b	12490441	0,00	32	32	115,31
Média		0,46	64,12	641,18	256,70

Tabela 5.7. Resultados do MBL (usando a busca local de Merz e Freisleben, 1999)

Exemplo	Melhor_Sol	Gap	It_Melhor	It_total	T_melhor
chr25a	3796	0,00	16	16	26,82
bur26a	5426670	0,00	0	0	1,21
nug30	6124	0,00	6	6	6,74
kra30a	88900	0,00	4	4	9,93
ste36a	9526	0,00	7	7	25,72
tai60a	7208572	0,85	25	28	270,36
tai80a	13557864	0,79	19	20	393,87
tai100a	21125314	0,72	15	15	529,37
sko100a	152002	0,15	15	15	526,17
tai60b	608215054	0,10	13	28	140,33
tai80b	818415043	0,18	16	20	330,26
tai100b	1185996137	0,43	14	15	490,34
wil100	273038	0,12	15	15	525,70
tho150	8133484	0,48	8	8	802,80
tai150b	498896643	1,48	8	8	803,81
lipa90a	360630	0,52	15	17	411,52
lipa90b	12490441	0,00	3	3	63,53
Média		0,34	11,71	13,24	315,20

Tabela 5.8. Resultados do MBT (usando como a busca local a Busca Tabu de Taillard, 1991).

Observe pelas tabelas 5.7 e 5.8, que a incorporação de Busca Tabu como a busca local dentro do algoritmo memético resultou em melhores resultados. Note também que o número total de iterações (gerações) no MBT é de 13,24 enquanto que no MBL é de 641,18, indicando que a Busca Tabu é um processo mais intenso e, conseqüentemente, mais eficaz, como mostram os resultados.

5.5. Comparação entre as meta-heurísticas

As Tabelas 5.9 e 5.10 reúnem os resultados obtidos pelas diferentes heurísticas. A Tabela 5.9 apresenta os valores de *gap* obtidos em cada exemplo, e a Tabela 5.10 os resultados médios.

Exemplo	BT	GA	MBL	MBT
chr25a	1,84	49,42	0,00	0,00
bur26a	0,00	0,26	0,08	0,00
cup30	0,00	4,18	0,07	0,00
kra30a	0,00	4,90	1,35	0,00
ste36a	0,00	14,59	0,00	0,00
tai60a	1,19	7,08	1,47	0,85
tai80a	0,88	6,48	1,01	0,79
tai100a	0,77	7,34	1,05	0,72
sko100a	0,07	6,39	0,14	0,15
tai60b	0,61	6,37	0,02	0,10
tai80b	3,39	10,08	0,57	0,18
tai100b	5,93	11,36	0,16	0,43
wil100	0,03	2,87	0,23	0,12
tho150	0,13	8,17	0,39	0,48
tai150b	2,39	11,19	0,80	1,48
lipa90a	0,50	1,06	0,47	0,52
lipa90b	0,00	24,41	0,00	0,00
Média	1,04	10,36	0,46	0,34

Tabela 5.9. GAPs obtidos por cada heurística para cada exemplo

Heurística	Gap	It_Melhor	It_total	T_melhor
BT	1,04	56488	132008	163,52
GA	10,36	11590	12836	356,54
MBL	0,46	64,12	641,18	256,70
MBT	0,34	11,71	13,24	315,20

Tabela 5.10. Resultados médios obtidos por cada heurística

Em termos médios a heurística MBT conseguiu obter os melhores resultados. Em alguns exemplos, a heurística MBL ou a BT conseguiu superar a MBT, mas nunca de maneira significativa. O GA se mostrou extremamente ineficaz quando utilizado sem uma busca local.

O algoritmo memético sendo utilizado juntamente com Busca Tabu já se mostrou eficaz na resolução de outros problemas, como visto em Berretta e Moscato, 1999.

6. CONCLUSÃO

Esse trabalho apresentou uma avaliação de algumas técnicas meta-heurísticas para a resolução do Problema de Designação Quadrática (QAP). Ainda não se conhece um algoritmo eficiente que apresente resolução exata desse problema, pois ele pertence à classe de problemas NP-difícil.

As meta-heurísticas implementadas foram Busca Tabu, Algoritmo Genético e Algoritmo Memético, sendo que este último foi implementado em duas versões, uma utilizando Busca Tabu e outra utilizando uma Busca Local, ambas encontradas na literatura.

O algoritmo de Busca Tabu implementado foi desenvolvido por Taillard, 1991. Um algoritmo memético implementado foi o desenvolvido por Merz e Freisleben, 1999. O outro foi idêntico, porém substituindo-se a Busca Local utilizada por Merz e Freisleben, 1999, pela Busca Tabu de Taillard, 1991. Além disso, foram realizados testes com o algoritmo memético sem nenhuma busca local, ou seja, um algoritmo genético.

As heurísticas foram testadas em um grupo de exemplos encontrados na literatura pesquisada. A Tabela abaixo apresenta os *gaps* médios obtidos por cada uma das heurísticas:

BT	GA	MBL	MBT
1,04	10,36	0,46	0,34

Pode-se atestar a superioridade do algoritmo memético para a resolução desse problema. Além disso, a utilização de Busca Tabu como a Busca Local dentro do Algoritmo Memético se mostrou a estratégia mais eficaz.

Como proposta pode-se considerar a implementação de elementos mais sofisticados nos algoritmos meméticos, como por exemplo, a estruturação da população como realizada em Berretta e Moscato, 1999. Também se pode testar outros operadores de cruzamento e mutação, e diferentes combinações de buscas locais no Algoritmo Memético.

7. REFERÊNCIAS BIBLIOGRÁFICAS

- Beasley, J. E., 1993, Lagrangean Relaxation, em *Modern heuristic techniques for combinatorial problems*, C.R.Reeves, ed, 243-303, Blackwell Scientific Publications.
- Berretta, R. e Moscato, P., 1999, The number partitioning problem: An open challenge for evolutionary computation?, em *New Ideas in Optimization*, D. Corne, F. Glover, e M. Dorigo, ed, McGraw-Hill.
- Burkard, R. E. e Çela, E., 1997, Quadratic and Three-Dimensional Assignments: An Annotated Bibliography, *Annotated Bibliographies in Combinatorial Optimization*, M. Dell'Amico, F. Maffioli e S.Martello, eds., Wiley, Chichester, pp. 373-392.
- Dell'Amico M., Maffioli, F. e Martello, S., 1997, *Annotated Bibliographies in Combinatorial Optimization*, Wiley, Chichester.
- Diaz, A., Glover, F., Ghaziri, H. M., Velarde, J. L. G., Laguna, M., Moscato, P. e Tsen, F. T., 1996, *Optimización Heurística y Redes Neuronales*, Editorial Paraninfo, Espanha.
- Elshafei, 1977, *Hospital Layout as a Quadratic Assignment Problem*, *Operations Research Quarterly*, vol. 28, no. 1, 167-179.
- Garey, M. R. e Johnson, D. S., 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman.
- Gero, J. S., Kazakov, V. and Schnier, T., 1997, Genetic engineering and design problems, in D. Dasgupta and Z. Michalewicz (eds), *Evolutionary Algorithms in Engineering Applications*, Springer Verlag, Berlin, pp.47-68.
- Glover, F. e Laguna, M. 1997, *Tabu Search*, Klumer Academic Publishers Norwell, Massachussets.
- Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading.
- Holland, J. H., 1975, *Adaption in Natural and Artificial Systems, An introductory analysis with application to biology, control, and artificial intelligence*, The University of Michigan Press, Ann Arbor, USA.
- Jünger, M., Martin, A., Reinelt, G. e Weismantel, R., 1994, Quadratic 0/1-Optimization and a Decomposition Approach for the Placement of Electronic Circuits , *Mathematical Programming*, 63, 257-280.

- Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I. e Dizdarevic, S., 1999, Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review* 13(2): 129-170.
- Li, Y., Pardalos, P.M., Ramakrishnan, K.G. e Resende, M.G.C., 1994, Lower bounds for the quadratic assignment problem, *Annals of Operations Research*, 50, pp 387-411.
- MAs Home Page, 2002, http://www.densis.fee.unicamp.br/~moscato/memetic_home.html
- Merz, P. e Freisleben, B., 1997, A Genetic Local Search Approach to the Quadratic Assignment Problem, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA'97)*, Thomas Bäck, ed., Morgan Kaufmann, pp. 465-472.
- Merz, P. e Freisleben, B., 1999, A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem, *Proceedings of the 1999 International Congress of Evolutionary Computation (CEC'99)*, IEEE Press, pp. 2063-2070.
- Merz, P. e Freisleben, B., 2000, Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem, *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 4, pp. 337-352, 2000.
- Merz, P. e Freisleben, B., 1996a, A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems, *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, IEEE Press, pp. 616-621.
- Merz, P. e Freisleben, B., 1996b, New Genetic Local Search Operators for the Traveling Salesman Problem, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature - PPSN IV*, H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel, eds., Volume 1141 of *Lecture Notes in Computer Science*, Springer, pp. 890-899.
- Moscato, P. e Norman, M., 1989, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Caltech Concurrent Computation Program, Report. 790.
- Moscato, P., 1989, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, P. Moscato, Caltech Concurrent Computation Program, C3P Report 826.

- Moscato, P., 1999, Memetic Algorithms: a short introduction, capítulo 14 em *New Ideas in Optimisation*, D. Corne, F. Glover, and M. Dorigo, eds., McGraw-Hill.
- Nemhauser, G. L. e Wolsey, L. A., 1988, *Integer and Combinatorial Optimization*. John Wiley & Sons.
- Oliver, I., Smith, D., e Holland, J., 1987, A study of permutation crossover operators on the traveling salesman problem. In Grefenstette, J. J., editor, *Genetic Algorithms And Their Applications: Proceedings of the Second Internal Conference on Genetic Algorithms*, 224-230, Hillsdale, New Jersey, Hove and London. Lawrence Erlbaum Associates, Publishers.
- Reeves, C., 1993, *Modern heuristic techniques for combinatorial problems*, John Wiley & Sons, Inc.
- Shorin-Kapov, J., 1994, Extensions of Tabu Search Adaptation to the Quadratic Assignment Problem, *Computers and Operations Research*, Vol.21(8), 855-865.
- Taillard, E., 1991, Robust Taboo Search for the Quadratic Assignment Problem, *Parallel Computing*, 17, pp. 443--445.
- Tate, D. M. e Smith, A. E., 1995, A genetic approach to the quadratic assignment problem, *Computers and Operations Research*, vol. 22, no.1, 73-83.
- Wilhelm, M. e Ward, T., 1987, *Solving quadratic assignment problems by simulated annealing*, IIE Transactions, 19:107-119, 15.