Recursos Educacionais / Educational Resources

Otimização Inteira:
Taxonomia das Meta-heurísticas

Luís Cavique
DCeT, Univ. Aberta
Luis.Cavique@uab.pt

Lisboa, dezembro 2020

Resumo

Este documento pretende complementar a bibliografia da UC de Otimização I oferecida no 1º semestre do 1º ano, no Doutoramento em Matemática Aplicada e Modelação (DMAM).

Nesta abordagem, depois de uma introdução aos problemas NP e aos limites inferiores e superiores de um problema de otimização apresenta-se uma taxonomia das meta-heurísticas (ME) e uma versão unificadora das meta-heurísticas híbridas.

A taxonomia considera as ME Intensivas que se concentram em regiões específicas do espaço de soluções e as ME Extensivas que se estendem pelo espaço de soluções.

São abordadas as ME intensivas: Simulated Annealing, Procura Tabu, Procura com Variação de Vizinhança e as ME extensivas: Algoritmos Genéticos, Scatter Search, Algoritmos Meméticos, GRASP, Colónias Formigas e Vocabulary Building.

Este trabalho tem como base o capítulo 3 da tese de doutoramento "Meta-heurísticas na Resolução do Problema da Clique Máxima e Aplicação na Determinação do Cabaz de Compras" de Luís Cavique, 2002.

Capítulo 3 da tese de doutoramento “Meta-heurísticas na Resolução do Problema da Clique Máxima e Aplicação na Determinação do Cabaz de Compras” de Luís Cavique, 2002.

Taxonomia das Meta-heurísticas (ME)

Neste capítulo apresenta-se uma taxonomia para as Meta-heurística (ME) que pretende sistematizar um conjunto de ME mais divulgadas. Para cada sub-grupo de ME são detalhados os seus procedimentos. Para sintetizar o conhecimento em meta-heurísticas desenvolvem-se dois operadores para ME híbridas.

3.1 – Introdução

3.1.1- Problemas de classe NP

A complexidade temporal de um algoritmo exprime que um algoritmo pode resolver um determinado problema em tempo O(f(n)), em que *n* representa a dimensão do problema, i.e., o número de elementos de entrada no algoritmo que é dependente de cada exemplo do problema. A função f(n) representa o número de vezes que um determinado processo é executado, como por exemplo, o número de comparações num algoritmo de ordenação.

Um algoritmo eficiente é aquele cuja complexidade é uma função polinomial dos dados de entrada. São exemplo disso, os algoritmos de busca de dados com complexidades variáveis como: O(1), O(log(n)), O(n); os algoritmos de ordenação de dados com complexidades: O(n.log(n)), $O(n^2)$; ou mesmo algoritmos com complexidade $O(n^5)$ são classificados como eficientes.

Por outro lado, complexidades algorítmicas como $O(2^n)$ ou O(n!) correspondem a algoritmos não eficientes. Os algoritmos de complexidade exponencial correspondem a problemas intratáveis, pois poderiam levar séculos a processar os dados de entrada

de um problema de dimensão média ou mesmo mais tempo que a idade do universo para um problema de grandes dimensões.

Em Optimização Combinatória é costume falar-se em problemas "fáceis" e problemas "difíceis". Para os primeiros, também conhecidos como problemas da classe P, existe um algoritmo eficiente para a sua resolução. Para os segundos não é conhecido nenhum algoritmo determinístico eficiente que os resolva em tempo útil, sendo por isso classificados na classe NP (não-determinista em tempo polinomial) [Garey e Johnson 1979].

Existe outra classe de problemas equivalentes entre si que correspondem aos de "maior dificuldade" dentro de todos os problemas NP. Os problemas dessa nova classe são denominados de NP-completos. Estes problemas são conhecidos como problemas "fáceis de definir e difíceis de resolver". Os problemas classificados como NP-completos podem ser transformados (em tempo polinomial) uns nos outros. Se, para um deles, se encontrar um algoritmo polinomial então, é evidente que todos eles poderão ser resolvidos em tempo polinomial.

Exemplos de problemas de classe P em optimização combinatória, são o problema do caminho mais curto, o problema da árvore de suporte de custo mínimo ou o problema do fluxo máximo.

Como exemplos de problemas da classe NP-completos, podemos referir o problema da clique máxima, o problema da cobertura por vértices, o problema do conjunto de independência de vértices e outros, mais conhecidos, como o problema do caixeiro viajante ou o problema da mochila.

Existem ainda pares de problemas cuja formulação é muito semelhante, no entanto um deles pertence à classe P e outro à classe NP. Estamos a referir-nos, por exemplo, ao problema do caminho mais curto que é da classe P sendo o problema do caminho mais longo da classe NP-completo, ou ao problema da cobertura por vértices que é um problema NP-completo e o problema da cobertura por arestas que é um problema da

classe P. Pode parecer uma contradição, mas o que acontece é que alguns destes problemas possuem uma estrutura tão específica que é possível encontrar um algoritmo eficiente para os resolver.

Problemas de decisão e optimização

Na Teoria da Complexidade existem duas formas de especificar os problemas: os problemas de decisão e os problemas de optimização.

Num problema de decisão pretende-se encontrar um sub-conjunto de dados que satisfaça um objectivo específico. Por exemplo, no caso do Problema da Clique Máxima, temos:

Dados: Um grafo G e um inteiro k >0

Objectivo: A clique máxima possui k vértices?

A resposta a um problema de decisão é simplesmente "sim" ou "não". Num problema de optimização pretende-se encontrar, de entre todos os sub-conjuntos de dados, aquele que satisfaz o objectivo especificado. No mesmo exemplo do Problema da Clique Máxima, temos:

Dados: Um grafo G

Objectivo: Encontrar em G uma clique máxima.

O problema de optimização diz-se NP-difícil se está relacionado com um problema de decisão NP-completo. É usual dizer que "este problema (de optimização) é NP-completo", evidentemente o que se está a querer dizer, apesar da imprecisão de linguagem, é que "a versão de decisão do problema de optimização é NP-completo". Na prática, parece preferível utilizar o termo NP-difícil para descrever o problema de optimização.

3.1.2 - Limites superiores e inferiores da solução óptima

Para obter a solução óptima de um problema os métodos exactos baseiam-se em técnicas de enumeração exaustiva. São denominados por métodos de pesquisa em árvore, conhecidos por 'branch-and-bound' na área de investigação operacional e A* na área de inteligência artificial. Estes métodos consideram uma separação do conjunto de soluções admissíveis do problema inicial e, seguidamente, resolvem cada um dos problemas parciais obtidos. Cada nó da árvore corresponde a um atributo do problema e as ramificações correspondem a partições do conjunto de soluções admissíveis. Às folhas da árvore, ou nós terminais, irão corresponder as soluções completas.

Os algoritmos exactos, apesar de garantirem a obtenção da solução óptima, encontram-se, na maior parte das situações reais, associados a um grande esforço computacional, dado o elevado número de soluções a analisar.

Os algoritmos aproximados foram desenvolvidos para dar resposta à impossibilidade de resolver uma grande variedade de problemas de optimização. Muito frequentemente, quando procuramos a resolução de um problema, somos confrontados com o facto do problema ser NP-difícil. Nas palavras de Garey e Jonhson: 'Não consigo encontrar um algoritmo eficiente para este problema, mas também, nenhum destes ilustres senhores o consegue' [Garey e Johnson 1979].

Se a solução óptima é difícil de encontrar, é razoável sacrificar a optimalidade e procurar uma boa solução que possa ser processada de forma eficiente. Claro que iremos sacrificar a optimalidade o mínimo possível, ganhando o máximo na eficiência. O balanço entre a optimalidade e a tratabilidade é o paradigma dos algoritmos aproximados.

O principal tema deste trabalho desenvolve-se à volta dos referidos algoritmos e da proximidade do óptimo alcançado em tempo polinomial. Na figura 9 apresentam-se os limites superiores e inferiores num problema de maximização. A avaliação das

heurísticas de aproximação ao óptimo são importantes para alcançar os limites inferiores num problema de maximização. Os limites inferiores apresentados por ordem crescente de qualidade, i.e., por grau de proximidade da solução óptima, são as soluções das heurísticas construtivas, das heurísticas de melhoramento e das meta-heurísticas.

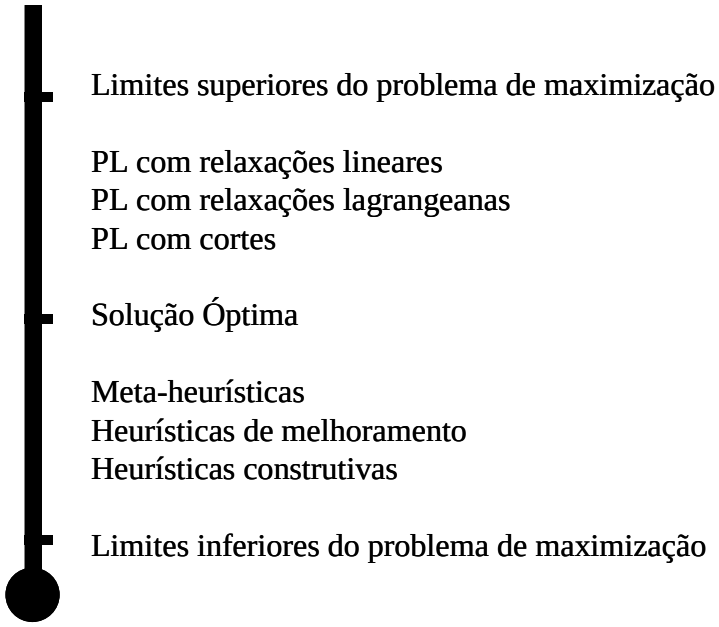

Figura 1 – Limites inferiores e superiores do problema de maximização

Por outro lado, os métodos que devolvem soluções não admissíveis provenientes da resolução de problemas que sofreram relaxações, são também conhecidos como heurísticas de 2ª ordem.

A solução óptima está contida no intervalo entre o limite superior e inferior. Para um problemas de maximização, os limites inferiores da solução óptima (Opt) são dados por heurísticas (He) e os limites superiores por relaxações (Rel).

$$He <= Opt <= Rel$$

Para medir a qualidade das solução obtidas, são utilizados os desvios (“gap”) relativos percentuais.

$$gap = [(Rel - Opt)/ Opt]x\ 100 \quad e \quad gap = [(Opt - He)/ Opt]x\ 100$$

Em geral, como não é conhecida a solução óptima para o problema e portanto não é conhecido o valor de Opt, utiliza-se um majorante do erro percentual.

$$gap = [(Rel - He)/ He]x\ 100$$

Quanto menor for a diferença entre os limites (“gap”), maior é a qualidade das soluções.

Métodos de Aproximação

O termo Heurística é usado para referir um conjunto de passos ou um conjunto de regras práticas, que têm como objectivo resolver um problema em tempo útil. Uma Heurística distingue-se de Algoritmo por não garantir a optimalidade nem eventualmente a admissibilidade. O termo Heurística tem origem na palavra grega “heuriskein” que significa encontrar ou descobrir. (Arquimedes terá dito “heureka” (encontrei / descobri) ao descobrir o princípio da impulsão da água.)

Nesta fase, pretendemos definir os conceitos de solução completa, solução parcial e elementos (ou atributos) de uma solução, em problemas de optimização combinatória. Uma solução é composta por um conjunto de elementos ou atributos; por exemplo no caso do PCM, uma solução corresponde a um conjunto de vértices de um grafo. Define-se solução parcial a solução que permite a inclusão novos elementos através de um processo heurístico; no caso do PCM, uma clique não-maximal é uma solução parcial porque permite a inclusão de mais vértices. Define-se como solução completa uma solução que não permite a inclusão de novos elementos; ainda no caso do PMC, uma clique maximal é uma solução completa.

Os métodos aproximados abrangem uma série de métodos heurísticos em que a optimalidade não é garantida mas o tempo computacional é bastante aceitável, permitindo em geral a obtenção de soluções admissíveis. Iremos distinguir, em seguida, as Heurísticas Construtivas, as Heurísticas de Melhoramento e as Meta-heurísticas.

As Heurísticas Construtivas, também denominadas Greedy ou Gulosas, partem de uma solução vazia e vão sucessivamente inserindo novos atributos de acordo com um critério de optimização, até que a solução se torne completa. Não havendo possibilidade de incluir novos elementos, o procedimento pára e devolve a solução completa.

As Heurísticas de Melhoramento partem de uma solução admissível e com base em alterações da solução, como, por exemplo, retirar e/ou inserir elementos, procuram melhorar a qualidade da mesma. Um exemplo bem conhecido das heurísticas de melhoramentos é a heurística 2-optimal [Lin 1965] para o problema do caixeiro viajante.

A principal dificuldade dos algoritmos aproximados reside no facto que a maior parte dos problemas combinatórios possuírem alguns óptimos globais e existirem muitos óptimos locais não tendo as heurísticas capacidade de distinguir um óptimo local de um global.

A partir do início dos anos 1980s surgem novos métodos heurísticos, apelidados de meta-heurísticas por Glover [1986], mas também conhecidos por Heurísticas Modernas. A rápida divulgação destes métodos deve-se à sua capacidade de adaptação a problemas de grande dimensão, como são a maior parte dos problemas reais. Uma abordagem introdutória pode ser encontrada em Pirlot [1996]. Em Reeves [1995] são detalhadas algumas das meta-heurísticas mais conhecidas, sem a preocupação de as integrar.

Meta-heurísticas são heurísticas de âmbito geral que possuem as seguintes características: são processos iterativos com uma ou mais soluções completas com complexidade algorítmicas baixas, tanto relativamente ao tempo como ao espaço, que têm em vista o melhoramento das soluções utilizando novas combinações no espaço inter e intra-soluções, evitando a prisão em óptimos locais (como acontece com as heurísticas construtivas e de melhoramento local). As meta-heurísticas distinguem-se das restantes heurísticas dedicadas a problemas específicos, porque incorporam estratégias de carácter muito geral que as tornam independentes dos problemas.

Uma tentativa de unificar as meta-heurísticas a Programação com Memória Adaptável [Taillard et al. 1998] integra o Simulated Annealing, a Procura Tabu, os Algoritmos Genéticos, o Scatter Search e o Algoritmo da Colónia de Formigas.

Em seguida apresenta-se um procedimento geral para qualquer meta-heurística, baseado no algoritmo de Programação com Memória Adaptável onde:

- S sub-conjunto, não vazio, de soluções completas
- S´ sub-conjunto de soluções completas em fase de teste ou provisórias

Procedimento Geral de Meta-heurísticas

1. Iniciar a Memória;
2. Enquanto não encontrar a condição de fim
    2.1. Como base nas soluções em Memória, encontrar S;
    2.2. Melhorar ou recuperar S obtendo S';
    2.3.Actualizar a Memória com S';
3. Fim ciclo;

O procedimento "Meta-meta-heurístico", é utilizado no conjunto de todas meta-heurísticas. Dado um conjunto de soluções S existente em memória, em cada iteração S é transformado num conjunto de soluções provisórias S', que irá posteriormente actualizar a memória.

Trabalhar com meta-heurísticas onde é possível a divisão do espaço de soluções, facilita a implementação de programas com computação paralela. A utilização destas técnicas para problemas que são utilizados com pouca frequência tem uma utilidade discutível, contudo para problemas em tempo real, o recurso à implementação dos programas em paralelo torna-se muito adequada.

Em ambientes onde o problema é alterado várias vezes, havendo a necessidade de desenvolver algoritmos reactivos que apresentem soluções a qualquer instante ("anytime algorithms"), as meta-heurísticas têm grande aceitação. Ao encontrar melhor soluções, a meta-heurística transmite essa informação ao sistema real, orientando-o para a melhoria contínua.

Métodos de Relaxação do Problema

As relaxações de restrições em problemas formulados com programação linear, ou heurísticas de 2ª ordem, são desenvolvidas para obter soluções óptimas mas não admissíveis para um problema específico, permitindo obter limites superiores para um problema de maximização.

Na Figura 1 apresentam-se os limites superiores e inferiores num problema de maximização. A avaliação dos métodos de relaxação do problema em programação linear são importantes para alcançar os limites superiores num problema de maximização. Os limites superiores apresentados por ordem crescente de qualidade, i.e., por grau de proximidade da solução óptima, são as relaxações lineares, relaxações lagrangeanas e métodos de corte.

Na relaxação linear, dada uma formulação em programação linear inteira (ou mista) do problema, irá relaxar a integralidade das variáveis, da qual resulta um problema de programação linear.

Outro método muito utilizado é a relaxação langrangeana. Neste método é necessário definir qual o subconjunto de restrições a relaxar e incorporar estas restrições na função objectivo associado a um conjunto de valores reais, designados por

multiplicadores de Lagrange. Para a optimização de uma relaxação lagrangeana é geralmente utilizado o método do subgradiente. Este método efectua a alteração sucessiva do valor dos multiplicadores, ao longo de várias iterações, com vista a optimizar a função objectivo.

3.1.3 - Taxonomia das Meta-heurísticas

Nesta secção vamos resumir brevemente as características das várias meta-heurísticas existentes, apresentando uma nova taxonomia das meta-heurísticas. Propomos uma divisão das Meta-heurísticas em dois grandes grupos: Intensivas e Extensivas.

As meta-heurísticas intensivas acumulam esforços em regiões específicas do espaço de soluções, actuando em profundidade. Por outro lado as meta-heurística extensivas, que agrupam os algoritmos evolutivos e as heurísticas de multi-partida, tem a característica de se estenderem por várias regiões no espaço de soluções.

Tabela 1 – Taxonomia das Meta-heurísticas

<table>
<tr><th colspan="2">Características</th><th>Meta-heurística</th></tr>
<tr><td colspan="2">Concentram-se em regiões específicas do espaço de soluções</td><td>ME Intensivas</td></tr>
<tr><td rowspan="2">Estendem-se pelo espaço de soluções</td><td>Populações de soluções</td><td>ME Extensivas com Populações</td></tr>
<tr><td>Solução única</td><td>ME Extensivas de Amostragem</td></tr>
</table>

Chamam-se Meta-Heurísticas Intensivas às meta-heurísticas baseadas em procura local ou de procura adaptável. As Meta-Heurísticas Intensivas exigem a definição das

estruturas de vizinhança e um conhecimento profundo do problema, apresentando capacidades na intensificação da pesquisa em regiões específicas.

Chamam-se Meta-Heurísticas Extensivas com Populações à classe das meta-heurísticas que incluem os algoritmos evolutivos. As meta-heurísticas baseadas em populações de soluções têm a vantagem de ter uma maior independência relativamente ao problema, não necessitando de conhecer qualquer das suas idiossincrasias. Estes algoritmos estendem-se pelo espaço de soluções possíveis numa fase inicial e com o desenrolar do processo procuram convergir para a solução óptima. As técnicas mais divulgada são os Algoritmos Genéticos, as Estratégias Evolutivas e o Scatter Search.

Chamam-se Meta-Heurísticas Extensivas de Amostragem aquelas que se dispersam pelo espaço de soluções mas que utilizam uma única solução de cada vez, gerando uma amostra de soluções. Refira-se que estas meta-heurísticas são as mais antigas de todas as meta-heurísticas, sendo baseadas na repetição de uma qualquer técnica de pesquisa com soluções de partida diferentes ("multi-start"). Técnicas de pesquisa como o GRASP desenvolvem esta estratégia com muito êxito.

3.2 – ME Intensivas

As Meta-heurísticas Intensivas baseiam-se essencialmente em técnicas de procura local onde a solução S_{n+1} depende da solução anterior S_n. Para introduzir o tema apresentamos o algoritmo descendente, seguido-se algumas das meta-heurísticas mais recentes: o Simulated Annealing, a Procura Tabu e a Procura com Variação da Vizinhança.

3.2.1 – Introdução

A procura local consiste na passagem de uma solução S_n para uma outra solução S_{n+1} na vizinhança da primeira, de acordo com determinadas regras, no espaço de soluções admissíveis X. A sequência de soluções é denominada trajectória no espaço de

soluções. Definindo por $S_n \oplus m$ o movimento que produz a transição de S_n para a uma outra solução S_{n+1}, a estrutura de vizinhança de $V(S_n)$ pode ser expressa da seguinte forma: $V(S) = \{ S_{n+1}: S_{n+1} = S_n \oplus m, \forall\ S_n \oplus m \in X\}$

A procura local é iniciada num ponto qualquer $S_n \in X$ e a nova solução S_{n+1} é escolhida na vizinhança $V(S_n)$. Isto implica associar a cada $S \in X$ uma vizinhança $V(S) \subseteq X$. A forma mais comum de encontrar a nova solução S_{n+1} é escolher a melhor do conjunto de soluções de $V(S_n)$. Num problema de maximização poder-se-ia escolher para S_{n+1} qualquer solução de $V(S_n)$ tal que $f(S_{n+1}) > f(S_n)$, onde *f* representa a função de avaliação das soluções. Seja S_{n+1} a solução de V(S) onde *f* toma o valor máximo. Essa é a melhor solução de V(S) e se $f(S_{n+1}) > f(S_n)$ então a nova solução corrente passa a ser S_{n+1}. Se não existir nenhuma solução em V(S) que melhore a solução S_n, então o processo pára. Esta estratégia é vulgarmente conhecida por Algoritmo Ascendente ou Subida da Encosta ('hill climbing').

Em seguida apresentamos o procedimento para o Algoritmo Ascendente ou de Subida da Encosta num problema de maximização, onde:

- S é a solução corrente
- S* é a melhor solução
- f é a função avaliação da solução
- V(S) é uma vizinhança de S

Algoritmo Ascendente ou Subida da Encosta

1. gerar S_0, $S^* = S_0$;
2. enquanto houver melhorias fazer:
   2.1. criar lista movimentos na V(S*);
   2.2. seja S a melhor solução em V(S*);
   2.3. se f(S)> f(S*) então S* = S;
3. fim ciclo;
4. devolver a melhor solução S*;

Um exemplo clássico deste algoritmo, para um problema de minimização, é o caso da heurística 2–optimal para o problema do caixeiro viajante, onde para dois pares de vértices não consecutivos se criam dois novos pares, através da troca dos arcos.

Na Figura 2, representa-se a trajectória de soluções, onde partindo de S_0 se identifica a solução S_n como sendo a solução óptima. A vizinhança da solução V(Sn) não inclui nenhuma solução com um valor da função objectivo superior, contudo a solução S_n não passa de um óptimo local.

É de notar que a escolha de uma boa estrutura de vizinhança é essencial para a eficácia do processo. Usando ainda o mesmo exemplo da Figura 2, é fácil verificar que se redefinisse uma vizinhança mais alargada seria possível continuar a pesquisa, levando a trajectória ao óptimo global.

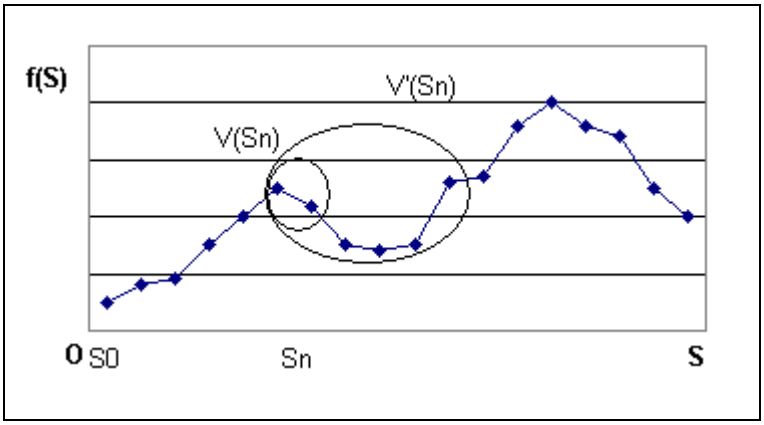

Figura 2 - Exemplo de prisão num óptimo local

Como não é possível saber à partida qual a estrutura de vizinhança, V(S), mais adequada, esta é a principal desvantagem do algoritmo ascendente, isto é, a incapacidade de fugir a óptimos locais. As meta-heurísticas como o Simulated Annealing e a Procura Tabu foram especialmente concebidas para evitar esta situação, utilizando uma degradação temporária da função de avaliação f(S). Por outro lado, a Procura com Variação da Vizinhança utiliza o alargamento da estrutura de vizinhança para permitir continuar a pesquisa, evitando também os óptimos locais.

3.2.2- Simulated Annealing

O Simulated Annealing é um algoritmo de optimização combinatória, que se baseia em certos princípios da termodinâmica; pode ser traduzido como "arrefecimento simulado" ou "simulação da têmpera". As ideias iniciais têm origem no algoritmo de Metropolis et al. [1953] que simula as variações de energia de um sistema quando sujeito a um processo de arrefecimento, até convergir para o estado "gelado". Trinta anos depois, Kirkpatrick et al. [1983] sugerem que este tipo de abordagem seja utilizado em optimização combinatória, fazendo o algoritmo convergir para a solução óptima.

É sabido que a formação de grandes cristais têm origem num arrefecimento muito lento e , pelo contrário, o arrefecimento brusco dá origem a cristais com imperfeições. Inicialmente, os movimentos das partículas são grandes e frequentes resultantes dos altos níveis energéticos, decrescendo à medida que a temperatura desce. Tal como na formação de cristais, o algoritmo de optimização com altas temperaturas tem uma "vizinhança" alargada; com a descida de temperatura a "vizinhança" tem tendência a diminuir.

A grande inovação neste algoritmo, relativamente ao algoritmo ascendente, é permitir a degradação temporária da função objectivo. Se o nível energético for grande, a trajectória de soluções tem capacidade de sair de um óptimo local e continuar a pesquisa.

O decremento da temperatura ou plano de arrefecimento parte de uma alta temperatura t_{max} que se mantém constante, por patamares, ao longo de um número determinado de passos. Em cada patamar k, a temperatura é dada por $t^k=\alpha^k.t_{max}$ com $0\leq\alpha\leq1$. Em cada patamar de temperatura, o processo repete-se L vezes, sendo o número total de iterações igual a k.L. Como parâmetros a definir temos a temperatura inicial t_{max}, o número k de patamares de temperatura, o comprimento L de cada patamar e a taxa de arrefecimento α.

Em cada iteração se a nova solução for melhor que a solução corrente, a solução corrente é substituída. Caso contrário, para a aceitação da nova solução, a degradação da função objectivo, vai depender de um determinado processo aleatório. A probabilidade de aceitação de uma nova solução é dada por exp (-d/t). Para altas temperaturas a probabilidade de aceitação exp (-d/t) é muito próxima de 1, desta forma, o algoritmo comporta-se como um algoritmo aleatório, pesquisando o espaço de soluções em passos largos. Para valores da temperatura perto de zero, a probabilidade de aceitação é baixa e o algoritmo comporta-se como um processo de melhoramento iterativo.

Em seguida apresentamos o procedimento de Simulated Annealing num problema de maximização, onde:

- f é a função avaliação da solução, S* é a melhor solução encontrada até ao momento, S é a solução corrente, S' é a próxima solução e V(S) é a vizinhança da solução S
- k é o número de patamares de temperatura e L é o número de iterações por patamar
- α é a taxa de arrefecimento e t^k é a temperatura em cada patamar k

Algoritmo Simulated Annealing

1. iniciar a solução S; fazer S* = S;
2. iniciar a temperatura t com um valor alto t_{max}, $t = t_{max}$;
3. repetir durante k patamares, até atingir o estado "gelado"
   3.1. repetir durante L passos
      3.1.1. gerar aleatoriamente uma solução $S' \in V(S)$;
      3.1.2. avaliar d = f(S) - f(S');
      3.1.3. se d < 0 então S = S';
      3.1.4. senão gerar um número aleatório p entre [0,1]
         3.1.4.1. se p < exp (-d/t) então S= S' ;
      3.1.5. se f(S) > f(S*) então S* = S;
   3.2. fim ciclo;
   3.3. decrementar a temperatura t de acordo com o arrefecimento definido;
4. fim ciclo;
5. devolver a melhor solução S*;

Num plano mais teórico, vários desenvolvimentos têm sido realizados. Batel-Anjo [1999] apresenta um modelo que garante a convergência assimptótica do Simulated Annealing; a partir do modelo são deduzidos algoritmos adaptáveis e gerados automaticamente.

3.2.3 - Procura Tabu

Tal como o Simulated Annealing, a Procura Tabu é uma meta-heurística que guia a procura para além dos óptimos locais. O método foi introduzido por Glover [1989] e diferencia-se dos métodos com origem na Inteligência Artificial, como o Simulated Annealing e os Algoritmos Genéticos, por utilizar estrutura de memória – em vez de fazer uma abordagem sem-memória possui uma memória adaptável. Uma descrição detalhada do método é apresentada por Glover e Laguna [1997].

O método consiste num procedimento iterativo que substitui uma solução pela seguinte utilizando uma estrutura de vizinhança, percorrendo trajectórias por "boas" regiões do espaço de soluções. Ao implementar uma heurística Tabu devem ser considerados três aspectos: a função objectivo f(S), a estrutura de vizinhança V(S), e finalmente os tipos de memórias. As estruturas de memória geralmente utilizadas são a memória de curta duração, materializada na lista Tabu T e uma memória de longa duração (MLD) que regista a frequência dos movimentos ao longo de um conjunto de iterações.

A Procura Tabu permite a passagem por uma solução pior que a solução corrente. Para evitar que o processo entre em ciclo, repetindo o mesmo conjunto de soluções, é introduzido no algoritmo uma lista onde são registados os movimentos efectuados. A memória de curta duração ou lista tabu, de dimensão *n*, vulgarmente implementada como uma lista circular, contém o conjunto dos *n* movimentos realizados imediatamente antes, por forma a evitar os movimentos inversos, voltando o problema à sua forma inicial.

Os movimentos proibidos constam da lista tabu (“tabu tenure”). Os movimentos possíveis na vizinhança da solução corrente não são permitidos enquanto permanecerem na lista. Inicialmente várias aplicações do método utilizaram para a dimensão da lista o número "mágico" de 7(±2) posições, mas com o rápido desenvolvimento de várias aplicações tornou-se perceptível que a dimensão da lista deveria depender da dimensão do problema. Assim, é consensual que para um problema com *n* elementos se deverá ter uma lista de dimensão $\sqrt{n}$.

Em seguida, apresenta-se o procedimento geral de Procura Tabu num problema de maximização, onde:

- S* é a melhor solução encontrada até ao momento
- S é a solução corrente
- S' é a melhor solução da lista de movimentos
- f é a função avaliação da solução
- V(S) é a vizinhança da solução S
- T é a lista tabu

Procedimento Geral de Procura Tabu

1. gerar S; fazer S*= S; iniciar T;
2. enquanto não atingir a condição de fim
   2.1. criar lista movimentos candidatos em [V(S) - T];
   2.2. S’= melhor solução da lista de movimentos ;
   2.3. actualizar lista tabu T;
   2.4. actualizar S=S’;
   2.5. se f(S) > f(S*) então S*=S ;
3. fim de ciclo;
4. devolver a melhor solução S*;

A condição de fim pode resultar de se ter esgotado o número de iterações, ou de não existirem soluções na vizinhança de S, ou ainda de se ter alcançado a solução óptima.

Para melhorar o desempenho da Procura Tabu têm sido desenvolvidas várias técnicas tais como o Critério de Aspiração, a Memória de Longa Duração, a Estratégia de

Oscilação, as Cadeias de Ejecção e o Path Relinking, que passamos as descrever em seguida.

Critério de Aspiração

A posse de um elemento na lista tabu ("tabu tenure") vai criar um corte no espaço de procura. O critério de aspiração, não é mais do que uma tentativa de ultrapassar este corte. Assim, o critério de aspiração determina que se realize o movimento apesar do elemento pertencer à lista tabu. No critério de aspiração, Asp tem em geral um valor numérico próximo de f(S*), sendo o movimento aceite se f(S')> Asp, num problema de maximização.

Memória de Longa Duração

Complementando a memória de curta duração baseada nos movimentos mais recentes, é possível utilizar a memória de longa duração (MLD). A MLD guarda a informação dos movimentos realizados ao longo das diversas iterações, tornando possível a procura em novos espaços de soluções, permitindo a diversificação da pesquisa. A MLD é usualmente baseada na frequência dos movimentos de elementos de forma a penalizar a função objectivo.

A ideia de utilizar funções que perturbem a função objectivo aparece desenvolvida por Charon e Hudryu [1999] no método do ruído, onde a função objectivo é transformada com a introdução de uma variável aleatória. Com a utilização da MLD a função objectivo vai ser alterada de forma determinística.

À MLD associam-se em geral valores numéricos baseados na frequência dos movimentos dos elementos. Devemos distinguir dois tipos de frequência: a frequência "residente" e a frequência "transiente". A frequência residente está relacionada com o número de vezes que um elemento ocorre nas soluções, durante um conjunto de iterações. Por outro lado a frequência transiente relaciona-se com o

número de vezes que um elemento é manipulado por movimentos (inserção, remoção) durante um conjunto de iterações.

A Procura Tabu utilizando a memória de curta duração cria uma estratégia de curto prazo (ou intensificação), que alterna com a estratégia de longo prazo (ou diversificação) utilizando a MLD.

Estratégia de Oscilação

Chama-se estratégia de oscilação a uma técnica que pode ser usada como outra forma de alternar a intensificação e a diversificação na procura. Esta técnica pode ser implementada relaxando um conjunto de restrições e aplicando vizinhanças específicas. O objectivo é atravessar regiões de soluções não admissíveis durante um determinado período da pesquisa. Este método irá ser desenvolvido no capítulo 4 na aplicação de meta-heurística intensivas ao problema da clique máxima.

Cadeias de Ejecção

As cadeias de ejecção incorporam uma estratégia de oscilação implícita produzida pela sequência de movimentos de ejecção, originando soluções parcialmente incompletas. Basicamente as cadeias de ejecção funcionam do seguinte modo: o movimento de ejecção consiste na substituição do elemento ejectado por um novo elemento; cada transformação identifica um nível da cadeia de ejecção, onde se obtém uma solução provisória (ou de teste); em cada nível da cadeia l é produzido um movimento adicional que garanta a admissibilidade da solução; o espaço de pesquisa consiste na construção de L níveis; depois da execução da sequência de movimentos é escolhido o melhor nível l^*, que corresponde à melhor solução de teste encontrada.

Aplicações das cadeias de ejecção podem ser encontrada em Cao e Glover [1997], Rego e Roucairol [1996], Rego [1998 a], Rego [1998 b] e Cavique, Rego e Themido [1999].

Path Relinking

Uma extensão da Procura Tabu é o Path Relinking que explora as combinações na vizinhança de soluções de elite, criando caminhos ("path") entre elas. Dadas duas soluções, uma dita inicial e outra final, para gerar os referidos caminhos é necessário escolher os movimentos que permitam: partindo da solução inicial, incluir progressivamente novos elementos, até alcançar a solução final. A geração de mais que um caminho ("relink") no espaço de vizinhança entre duas soluções explica o nome do método, onde se cria um primeiro caminho e, em seguida, se criam novos caminhos diferentes, explorando desta forma a vizinhança de soluções.

A trajectória da solução tem origem e destino em regiões admissíveis, podendo atravessar regiões não admissíveis, sem risco de se "perder". A esta estratégia foi dado nome de passagem do túnel ("tunnelling") e pretende unir as regiões admissíveis que aparecem como ilhas no espaço de soluções.

3.2.4 - Procura com Variação da Vizinhança

Este método de pesquisa local consiste na escolha aleatória de soluções na vizinhança da solução corrente, com uma variação sistemática do espaço das sub-vizinhanças [Hansen e Mladenovic 1998].

Inicialmente é definido um conjunto de vizinhanças V^k, $k=1,\ldots, k_{max}$, que será utilizado no método de procura. O método consiste na procura de um óptimo local na vizinhança de uma solução. Logo que não haja mais soluções a explorar, é escolhida outra V^{k+1}, com vista a alargar a vizinhança da solução. O alargamento da vizinhança no problema do caixeiro viajante corresponde à passagem de trocas de k-optimal a (k+1)-optimal. No caso de um problema de programação inteira binário corresponde a aumentar o número de variáveis que se complementam de k para k+1.

Em seguida apresentamos o procedimento de Procura com Variação de Vizinhança num problema de maximização, onde:

- S* é a melhor solução
- S é a solução corrente
- S' é a próxima solução
- f é a função avaliação da solução
- k é o número de vizinhanças (1.. k_{max})
- $V^k(S)$ é a vizinhança k da solução S

Procedimento de Procura com Variação de Vizinhança

1. iniciar S ; fazer S*=S;
2. para k=1 até k_{max}
   2.1. enquanto existir S'$\in V^k(S)$
      2.1.1. escolher aleatoriamente S' $\in V^k(S)$;
      2.1.2. se f(S') > f(S) então S=S';
   2.2. fim ciclo;
   2.3. alargar a vizinhança $V^k(S)=V^{k+1}(S)$;
3. fim ciclo;
4. fazer S*=S;
5. devolver S*;

A Procura com Variação de Vizinhança vai alargando o espaço de vizinhança ao longo das iterações, enquanto que o Simulated Annealing vai reduzindo o espaço de vizinhança na tentativa de convergir para a solução óptima.

3.3 - ME Extensivas com Populações

Uma grande variedade de ME Extensivas com Populações, ou Algoritmos Evolutivos como são conhecidos na área da Inteligência Artificial, têm sido propostos na literatura. Os mais conhecidos são os Algoritmos Genéticos propostos por Holland [1975] nos E.U.A, e paralelamente, deste lado do Atlântico, na Alemanha Rechenberg [1973] e Schwefel [1981] propõem as Estratégias Evolutivas. Também

pela mesma altura Glover [1977] apresenta o Scatter Search. Mais tarde aparecem os Algoritmos Meméticos [Moscato 1989] como forma de optimizar agentes com estratégias de cooperação e competição. Alguns autores incluem ainda os Sistemas de Classificação [Goldberg 1989], a Programação Evolutiva [Fogel et al. 1966] e a Programação Genética [Koza 1992] como Algoritmos Evolutivos, embora estes algoritmos tenham menos aplicação no domínio da optimização.

3.3.1- Algoritmos Genéticos

Os Algoritmos Genéticos (AG) fazem parte de uma família de modelos computacionais inspirados no princípio da selecção natural de Darwin, onde uma população, constituída por soluções de um problema específico evolui como se tratasse de uma espécie biológica. A optimização com AG é baseada na hipótese que durante a evolução natural, as leis da hereditariedade irão produzir populações com indivíduos com um melhor desempenho. Os princípios básicos dos AG foram introduzidos por Holland [1975] e mais tarde divulgado em trabalhos como os de Goldberg [1989], Michalewicz [1996] e Michalewicz e Fogel [2000].

Os AG simulam uma população de indivíduos, onde cada um deles representa uma solução para o problema e possuindo uma determinada qualidade ou mérito ("fitness") dada pela função de avaliação. Aos indivíduos mais dotados são dadas mais oportunidades de reprodução através de cruzamentos com outros indivíduos. Daqui surgem os filhos que possuem características idênticas às dos pais. Os membros da população com menor qualidade têm probabilidades mais baixas de reprodução e acabam por desaparecer. Ao fim de um determinado número de gerações um conjunto de características torna-se mais evidente, convergindo a população para um óptimo local.

As fases críticas no desenvolvimento de um Algoritmo Genético passam pela codificação do indivíduo, criação da população inicial, definição da função avaliação, definição de esquemas de selecção, operadores de cruzamento, operadores de mutação e esquemas de substituição da população.

i) Codificação

Na representação de cada solução (indivíduo ou cromossoma) é geralmente utilizado um vector binário, S. A cada S_i também se chama gene. O conjunto de parâmetros S de um cromossoma em particular é apelidado de genótipo. O genótipo contem informação necessária para a construção do indivíduo, que é apelidado de fenótipo.

ii) Diversidade na população inicial

Na geração da solução inicial deve ser garantida a diversidade de soluções, bem como a garantia de que todos os genes estão representados na população.

iii) Função de avaliação

Dado um cromossoma, a função de avaliação (objectivo, mérito ou "fitness") retorna um valor numérico do mérito que é suposto ser proporcional à utilidade do indivíduo.

iii) Esquemas de selecção

A selecção consiste na determinação do número de vezes que um indivíduo é escolhido para a fase de cruzamento. A *pressão da selecção* depende do grau com os melhores indivíduos são escolhidos, e quanto maior é a pressão, maior é a selecção de indivíduos com maior função de mérito. Por convergência da população entende-se que a população é constituída por indivíduos muito idênticos. A taxa de convergência depende directamente da pressão da selecção. Existem dois tipos de esquemas de selecção: a selecção proporcional e a selecção ordinal. A selecção proporcional baseia-se na qualidade dos indivíduos da população e são exemplos a selecção proporcional (roleta) e a selecção universal estocástica. Por outro lado a selecção ordinal baseia-se na ordem relativa (‘ranking’) dos indivíduos e podem apresentar-se, como exemplos, a selecção por torneio e a selecção por truncatura. Os esquemas de selecção ordinal são geralmente preferidos aos proporcionais porque são mais equilibrados na pressão da selecção.

iv) Operadores de cruzamento

O operador de cruzamento clássico é o cruzamento simples com um ponto de cruzamento. Mais tarde, é proposto o cruzamento com *n* pontos que não é mais do que a generalização do cruzamento simples. O cruzamento uniforme leva o conceito de cruzamento mais além, favorecendo os genes dependentes mas que não se encontram juntos. No cruzamento aritmético dados dois ascendentes a1 e a2, utiliza-se uma combinação linear convexa para gerar dois descendentes d1 e d2, da seguinte forma: $d1=\lambda.a1+(1-\lambda).a2$ e $d2=\lambda .a2+(1-\lambda).a1$, com $\lambda \in [0,1]$.

v) Operadores de mutação

A mutação desempenha um papel secundário mas indispensável nos AG e corresponde à alteração aleatória de um ou mais genes. A mutação introduz a garantia contra a perda prematura de espaços de procura. A extensão do operador de mutação dá origem a funções de reparação da solução, que garantem a admissibilidade da solução gerada.

vi) Substituição

A substituição é talvez o mecanismo mais sensível nos AG e nem sempre devidamente identificado. No Algoritmo Genético Canónico, em cada iteração toda a população é substituída pela seguinte, no caso do algoritmo não convergir, existe o risco de perder as melhores soluções encontradas. Para evitar este problema, o método elitista, vai guardar um conjunto das melhores soluções para a iteração seguinte. A substituição pode ainda incluir a opção de introdução de novos elementos. O método de estado-estável (steady-state) consiste na substituição de um conjunto muito reduzido de indivíduos em cada iteração, mantendo a população estável. Neste método coloca-se também a questão de saber qual o conjunto de indivíduos que deve sair da população: retirar aleatoriamente um conjunto de indivíduos, retirar os piores ou retirar os ascendentes (ou indivíduos semelhantes). A substituição dos indivíduos pelos pais minimiza as alterações da população, mantendo a diversidade.

Em seguida apresentamos o procedimento geral de um Algoritmo Genético com os operadores de selecção, cruzamento e mutação, onde:

- P e P’ são as populações de soluções
- S* é a melhor solução.

Procedimento geral de um Algoritmo Genético

1. Iniciar a população P;
2. Enquanto não existir condição de fim
   2.1. avaliar o mérito de cada indivíduo de P;
   2.2. P'= selecção dos elementos de P;
   2.3. P'= cruzamento dos elementos de P';
   2.4. P' = mutação dos elementos de P';
   2.5. substituição de P pelos elementos de P';
3. fim ciclo;
4. devolver a melhor solução S* existente em P;

O critério de paragem do algoritmo irá recair num dos seguintes: i)quando a solução óptima foi atingida, ii)por limitação de tempo ou por atingir o número máximo de iterações, iii)quando não existe melhoria durante um conjunto de iterações, ou iv)quando se verifica a convergência da população.

Entende-se por "epitasis" a forma como a função objectivo varia quando uma variável que depende de outras variáveis é alterada. Nos AG define-se como "decepção" quando o algoritmo se distancia da solução óptima. A "decepção" está directamente relacionada com os efeitos da "epitasis", digamos que é a condição necessária mas não suficiente. Os AG não têm um desempenho eficaz quando as variáveis são muito inter-relacionadas.

Um problema clássico dos AG ocorre quando um conjunto de indivíduos domina rapidamente a população com as suas características, convergindo a solução para um óptimo local. A capacidade do AG de continuar a pesquisa de melhores soluções através do operador de cruzamento é eliminada. Só o operador de mutação terá

capacidade de continuar a pesquisa, aproximando o AG de um algoritmo de busca aleatória ("random search"). Na implementação de um AG, a convergência prematura é, sem dúvida, uma das principais preocupações, devendo ser dado especial atenção à falta de diversidade da população inicial e à pressão da selecção.

3.3.2- Estratégias Evolutivas

Enquanto que os AG foram desenvolvidos para resolver problemas de optimização discreta, as Estratégias Evolutivas (EE) foram inicialmente construídas para resolver problemas de optimização contínua. Por consequência, AG e EE diferem significativamente tanto na codificação como em outras estratégias de implementação. Contudo, utilizam os mesmos princípios da evolução natural: selecção, combinação e mutação.

As diferenças fundamentais entre as EE e os AG dizem respeito ao esquema de codificação, aos operadores genéticos e à estratégia de substituição. Dado que as EE optimizam funções contínuas, a codificação das variáveis é composta de números reais, permitindo a utilização de operadores múltiplos.

Relativamente à estratégia de substituição das populações existem dois tipos fundamentais de EE [Bäck 1994]: o EE(μ, λ) e o EE(μ+λ). O símbolo μ representa o número total de ascendentes e o símbolo λ o número de descendentes. No primeiro tipo, EE(μ, λ), os descendentes substituem os ascendentes como nos AG. No segundo tipo, EE(μ+λ), a população toma a sua dimensão original, escolhendo as melhores soluções entre ascendentes e descendentes.

3.3.3- Scatter Search

O Scatter Search foi introduzido por Glover [1977] no estudo de problemas de programação linear inteira e utiliza modelos espaciais; ao conjunto de boas soluções vectoriais chamamos Conjunto de Referência (CRef). Este método utiliza combinações lineares de soluções com vista a produzir soluções novas para as

gerações seguintes. Para além de utilizar combinações de soluções, o método inclui heurísticas de reparação e melhoramento das soluções de teste. As melhores soluções encontradas são incluídas no CRef.

O Scatter Search contrasta com os AG na representação das soluções ao utilizar representações vectoriais em vez de vectores binários. Para além disso utiliza *combinações estruturadas* de duas ou mais soluções das quais podem resultar diferentes soluções provisórias; as soluções de teste são melhoradas e no CRef pretende-se manter o equilíbrio entre a diversidade das soluções e a salvaguarda das melhores soluções.

A aplicação desta Meta-heurística Extensiva na resolução do problema da Clique Máxima será novamente apresentada e desenvolvida no capítulo 5.

3.3.4 - Algoritmos Meméticos

Uma meta-heurística pode ser classificada de Algoritmo Memético desde que seja baseada numa pesquisa com população e que recorra à pesquisa local para criar novas configurações das soluções.

O termo "memético" foi introduzido por Moscato [1989], sendo também conhecidos por Algoritmos Genéticos Hibrídos, embora a denominação de Algoritmo Memético enfatize mais a diferença com os Algoritmos Genético Canónicos [Coll, Durán e Moscato 1999].

Numa das implementações dos Algoritmos Meméticos descritas por Berretta e Moscato [1999] é apresentada uma população com um número fixo de 13 elementos organizados numa estrutura em árvore ternária com três níveis, hierarquizados pela qualidade das soluções. Esta estrutura pode ser interpretada como uma variante dos modelos das ilhas nos Algoritmos Genéticos, incluindo ainda a capacidade de estratificar as melhores soluções das soluções diversas.

3.4 - ME Extensivas com Amostragem

Os métodos de amostragem procuram encontrar conjuntos de soluções, ou amostras de soluções, no espaço de soluções admissíveis. As meta-heurísticas extensivas com amostragem trabalham com uma única solução em cada iteração. O objectivo é encontrar muitas soluções diferentes, espalhadas pelo o espaço de soluções, entre as quais se espera encontrar soluções de grande qualidade, conforme é esquematizado na Figura 3.

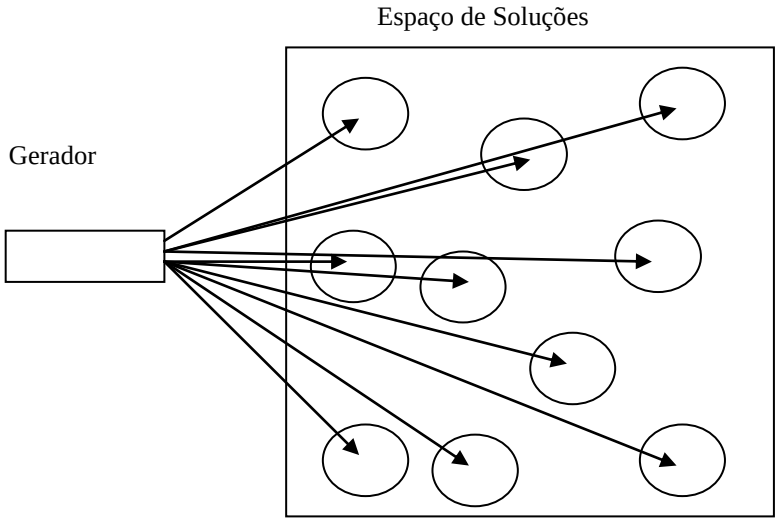

Figura 3 – Técnicas de Amostragem

3.4.1- O método Multi-partida

O método Multi-partida ("multi-start") ou de amostragem simples é talvez a meta-heurística mais antiga, que tem mostrado grande capacidade de adaptação a vários problemas com resultados satisfatórios. O método, de grande simplicidade, utiliza repetidamente diferentes soluções iniciais, que servem de entrada a uma heurística específica para o problema. As soluções iniciais podem ser geradas aleatoriamente ou através de processos sistemáticos.

Em seguida apresentamos o procedimento geral do algoritmo multi-partida num problema de maximização, onde:

- S é a solução corrente
- S* é a melhor solução
- f é a função avaliação da solução

Procedimento Multi-partida

1. iniciar S e S*;
2. enquanto não se verificar condição de fim
   2.1. gerar nova parcial S;
   2.2. transformar S numa solução completa;
   2.3. se f(S) > f(S*) então S* = S;
3. fim ciclo;
4. devolver S*;

Desenvolvida a heurística, o método de multi-partida é de fácil implementação. O procedimento consiste na repetição de uma heurística dedicada que transforma uma solução parcial numa solução completa. A nova solução de partida pode consistir num elemento, numa solução parcial ou ainda numa solução completa. A heurística específica pode ser gerada aleatoriamente, utilizar um processo construtivo ou ainda usar um procedimento melhorativo. O sucesso desta meta-heurística depende do equilíbrio entre três variáveis: as soluções de partida, a escolha da heurística dedicada e do número de repetições do procedimento. A dimensão da amostra de soluções é dada pelo número de repetições.

3.4.2- Meta-heurística GRASP

A meta-heurística "Greedy Randomized Adaptive Search Procedures" (GRASP) consiste na combinação de heurísticas construtivas e de procura local [Feo e Resende 1989]. O GRASP é uma concretização do procedimento multi-partida. Em cada iteração gera uma nova solução no espaço de soluções em duas fases: primeiro usa

uma heurística construtiva mista ou semi-gulosa, i.e. simultaneamente gulosa ("greedy") e aleatória, e a seguir uma heurística melhorativa baseada em procura local.

Em seguida apresentamos o procedimento de GRASP para um problema de maximização, onde:

- S é a solução corrente
- S' é a solução melhorada
- S* é a melhor solução
- f é a função avaliação da solução

Procedimento GRASP

1. iniciar S e S*;
2. repetir para um conjunto de iterações
   2.1. construir S usando um algoritmo semi-guloso;
   2.2. aplicar procura local usando S dando origem a S';
   2.3. se f(S') > f(S*) então S* = S';
3. fim ciclo;
4. devolver S*;

O algoritmo semi-guloso vai utilizar uma função g(x) para avaliar o impacto da solução parcial, dada pela união de um novo elemento com a solução corrente. A heurística é dita "adaptável" já que em cada iteração é reconhecido o benefício da inclusão de um elemento x na solução S. É utilizada uma lista restrita de elementos candidatos (LRC) com as melhores avaliações da função g(x). Por fim é escolhido aleatoriamente da lista LRC um novo elemento que irá integrar a solução.

Em seguida apresentamos o procedimento de Construtivo Semi-guloso para um problema de maximização, onde:

- S é a solução corrente
- x é um elemento da solução
- g é a função de avaliação gulosa
- LRC é a lista restrita de elementos candidatos
- max é o valor máximo de g(x)
- min é o valor mínimo de g(x)
- α é o parâmetro de controlo, $\alpha \in [0,1]$

Procedimento Construtivo Semi-guloso

1. iniciar S;
2. enquanto a solução S não for completa
   2.1. avaliar o benefício de inclui novos elementos com a função g(x);
   2.2. criar LRC={x: $g(x) \geq \max - \alpha$ (max-min) };
   2.3. escolher aleatoriamente de LRC um elemento x;
   2.4. $S = S \cup \{x\}$;
3. fim ciclo;
4. devolver S;

O procedimento mostra que o parâmetro α controla a quantidade de aleatoriedade e "gulodice" do algoritmo. Para α=0 está-se em presença de um procedimento "greedy" onde são escolhidos os elementos com benefício máximo, e quando α=1 corresponde a um algoritmo aleatório, com benefício superior ao valor mínimo de g(x).

A fase construtiva não encontra necessariamente um óptimo local pelo que se aplica um procedimento de procura local já desenvolvido na secções anteriores.

A meta-heurística GRASP apresenta-se de fácil implementação depois de definidos as heurísticas construtivas e de melhoramentos. A heurística construtiva tem a vantagem de ter reduzido o número de parâmetros, apenas são necessários o valor de α e o número de iterações.

3.4.3- Algoritmo das Colónias de Formigas

O algoritmo das colónias de formigas consiste na simulação do comportamento de um conjunto de agentes que interagem de uma forma bastante simples [Colorni, Dorigo e Maniezzo 1992].

As formigas são capazes the encontrar caminhos entre o formigueiro e a fonte de alimento e ultrapassar obstáculos com relativa facilidade. Elas tem a capacidade de comunicar utilizando uma substancia química chamada feromona, que é tanto mais evidente quanto maior for o número de formigas.

Como exemplo da aplicação do Algoritmo das Colónias de Formigas para o problema do Caixeiro Viajante temos:

i) cada formiga irá descrever o circuito do caixeiro viajante.

ii) cada arco (i,j) é escolhido com base numa regra aleatória entre as cidades não visitadas; a escolha de arco (i,j) é directamente proporcional à quantidade de feromona, $\Gamma(i,j)$, e inversamente proporcional ao custo do arco, c(i,j).

iii) depois do circuito estar concluído, para cada arco por onde a formiga passou, o valor da feromona irá ser incremento do valor δ, donde $\Delta\Gamma(i,j)= \Delta\Gamma(i,j)+\delta$.

iv) finalmente, para todos os arcos vai existir uma evaporação da feromona, dada por um factor de evaporação β, com $0<\beta<1$, de onde resulta $\Gamma(i,j)= \beta . \Gamma(i,j)+ \Delta\Gamma(i,j)$

Dado que os valores iniciais de $\Gamma(i,j)$ são iguais para todos os arcos, as formigas inicialmente fazem circuitos aleatórios. Se considerarmos várias formigas circulando no grafo em simultâneo, os arcos que têm maior probabilidade de serem escolhidos são os arcos com maior quantidade de feromona, já que foram considerados os arcos mais desejados por outras formigas.

Em seguida apresentamos o procedimento geral do Algoritmo das Colónias de Formigas para um problema de maximização, onde:

- S é a solução corrente
- S* é a melhor solução
- f é a função avaliação da solução
- x é o elemento da solução
- $\Gamma(x)$, $\Delta\Gamma(x)$ é a quantidade de feromona de x
- β é o factor de evaporação e δ é o incremento de feromona

Procedimento Algoritmo das Colónias de Formigas

1. iniciar S e S*;
2. enquanto não se verificar condição de fim
   2.1. para cada formiga
      2.1.1. construir uma solução S;
      2.1.2. actualizar a quantidade de feromona $\Delta\Gamma(x)= \Delta\Gamma(x)+\delta, \forall\ x \in S$;
      2.1.3. se f(S) > f(S*) então S* = S;
   2.2. fim ciclo;
   2.3. actualizar a quantidade de feromona $\Gamma(x)=\beta.\Gamma(x)+ \Delta\Gamma(x), \forall\ x$;
3. fim ciclo;
4. devolver S*;

Utilizando o efeito simultâneo de várias formigas, em que cada uma contribui para avaliar os elementos que pertencem à solução, os melhores elementos são os que receberem maior quantidade de feromona. A feromona pode ser considerada uma memória (de longa duração) do algoritmo, actualizada pela passagem de formigas e pela evaporação, registando o histórico de cada elemento.

O Algoritmo das Colónias de Formigas não reflecte o modelo natural. Trabalha com uma única solução em cada iteração em vezes de uma população e a comunicação entre agentes realiza-se através dos valores de feromona, não existindo comunicação directa.

3.4.4- Vocabulary Building

O procedimento de "Vocabulary Building" [Glover 1999] cria combinações estruturadas a partir do desmembramento da solução completa e da agregação de soluções parciais. O nome do procedimento vem da analogia com a evolução do vocabulário de uma linguagem natural, onde determinadas palavras caem de desuso ou adquirem novos significados e novas palavras são criadas através de combinações das anteriores.

Num primeiro passo, o procedimento decompõe a solução em soluções parciais e, num segundo passo, a partir das soluções parciais procura encontrar uma nova solução. O procedimento irá repetir dos processos de demolição e construção. A estratégia consiste em utilizar boas soluções parciais e procurar novos óptimos locais, orientando assim a direcção da pesquisa.

Em seguida apresentamos o procedimento de "Vocabulary Building" para um problema de maximização, onde:

- S é a solução corrente
- S* é a melhor solução
- f é a função avaliação da solução
- SP é o conjunto de soluções parciais

Procedimento de "Vocabulary Building"

1. iniciar S e S*;
2. agregar os elementos na solução completa S;
3. repetir um conjunto de iterações
   3.1. demolir: decompor S em soluções parciais SP;
   3.2. construir: partindo de SP encontrar uma nova solução S';
   3.3. actualizar S com S';
   3.4. se f(S) > f(S*) então S* = S;
4. fim ciclo;
5. devolver S*;

Variantes desta estratégia foram aplicadas a um conjunto de problemas nomeadamente ao problema da clique máxima e ao problema de sequenciamento de tripulações. No problema da clique máxima, desenvolve-se o conceito de fusão de duas cliques [Aggarwal, Orlin e Tai 1997] [Balas e Niehaus 1998] e a utilização de filtros sobre combinações de cliques [Cavique, Rego e Themido 2001 b]. No problema de sequenciamento de tripulações procura-se encontrar "o filho perfeito" na combinação de soluções [Lourenço, Paixão e Portugal 1998] e aplicar reduções à solução completa, servindo esta solução parcial para uma nova partida [Cavique 1994] [Cavique e Themido 1995].

3.5 – Heurística Híbridas

3.5.1- O Equilíbrio entre Estratégias Intensivas e Extensivas

Dado que as heurísticas para problemas combinatórios são muito dependentes da solução inicial, é possível que um algoritmo ao partir de uma solução inicial encontre a solução óptima em segundos e que ao partir de outra solução inicial possa demorar horas para encontrar a mesma solução. O tempo até encontrar a solução óptima apresenta uma grande variabilidade.

Em problemas combinatórios, o histograma das frequências dos tempos computacionais apresenta uma distribuição com pequenas variações na frequência que se prolonga com longas caudas (*heavy tails*), indicando que existem tantos casos extremos como casos médios, ao contrário da distribuição Normal com muitos casos médios e poucos casos extremos. Os estudos de Gomes et al. [1997] provaram que a cauda à direita pode ser muito longa, o que significa que existe um conjunto de soluções de partida que não levam na prática à solução óptima.

Conhecido este problema, uma forma de o ultrapassar este problema é fazer "rapid restarting", i.e. iniciar a pesquisa num outro ponto do espaço de soluções indo ao encontro de um nova solução de qualidade. O sucesso das estratégias de pesquisa

depende do equilíbrio entre a forma como se varre o espaço de soluções (estratégia extensiva) e a pesquisa local (estratégia intensiva). Para um intervalo de tempo limitado, são colocadas duas questões fundamentais: i) quantas vezes se repete o processo de pesquisa, na estratégia extensiva? ii) quando é que o processo da estratégia intensiva alcança a condição de fim?

Se por um lado as ME intensivas "puras" têm dificuldade em sair da região de pesquisa, as ME extensivas têm um comportamento deficiente quando as variáveis têm uma grande inter-dependência, com é referido nos fenómenos de "decepção" dos AG para problemas com alta "epitasis". Desta forma as soluções híbridas parecem uma forma de complementar as estratégias intensivas e extensivas.

Numa das mais recentes compilação de publicações de novas ideias em optimização de Corne, Dorigo e Glover [1999], as ME como as Colónias de Formigas, os Algoritmos Meméticos, o Scatter Search reflectem a tendência para a utilização de estratégias híbridas.

Para qualquer heurística é importante encontrar a heurística complementar: 'primais e duais', 'interiores e exteriores', 'construtivas e destrutivas', 'global e local', 'intensiva e extensiva', por forma a conjugar esses opostos.

Na análise de resultados computacionais de duas ou mais heurísticas é vulgar constatar que nenhuma das heurísticas domina ou é dominada pela outra. A expressão heurística dominada é utilizada no sentido da decisão multi-critério, a heurística X domina Y se e só se $f(X[i]) \geq f(Y[i]), \forall i$ e $\exists i: f(X[i]) > f(Y[i])$ sendo *i* um determinado exemplo de problema. A heurística X diz-se não dominada por Y, se não existir qualquer heurística Y que domine X. Por fim, duas heurísticas *não são comparáveis* se não se verificar a dominância de X por Y nem de Y por X.

Para heurísticas não comparáveis, julgamos do maior interesse a criação de heurísticas híbridas como forma de tirar partido da complementaridade das diferentes estratégias.

No capítulo 4, iremos pôr em prática esta metodologia com a Heurística Híbrida Primal-Dual.

3.5.2- Operadores de Híbridos

Por forma a modelar as arquitecturas de híbridos propomos a utilização de dois operadores: multiplicativo e aditivo, que passamos a explicar. O operador multiplicativo "*" é unário enquanto que o operador aditivo "+" é binário [Gendreau 1999].

Ao executar uma sequência de heurísticas, em que o resultado da primeira é a solução inicial da segunda, utilizamos um operador aditivo. Um exemplo bastante conhecido é o seguinte: dada uma solução inicial criada a partir de um heurística construtiva (HC) melhorar a solução com uma técnica de Procura Tabu (PT); representamos pela seguinte expressão: HC+PT.

Se existe uma repetição da heurística i.e. se a heurística é chamada um conjunto de vezes, utilizamos o operador multiplicativo. Por exemplo, a estratégia de multi-partida (MP) seguida de uma heurística (H) pode ser expressa da seguinte forma: *(MP+H).

A criação de híbridos pode ir tão longe quanto for necessário. Podemos associar Algoritmos Genéticos com Procura Tabu com uma Heurística Construtiva de diversas formas: utilizando o Algoritmo Genético (AG) como supervisor de uma pesquisa local, AG[*(HC+PT)] ou ainda utilizar uma heurística multi-partida ao nível superior e criar uma sequência de heurísticas onde o resultado de uma Procura Tabu é desenvolvido por um Algoritmo Genético que retorna a solução a uma nova Procura Tabu, expresso da seguinte forma: *(MP+HC+PT+AG+PT). Hierarquizando as meta-heurísticas temos: a um nível superior está com certeza a estratégia de multi-partida, a um nível intermédio as técnicas baseadas em populações e ao nível mais operacional, a pesquisa local, ou seja: MP+[AG*(HC+PT)].

Para além da criação dos operadores híbridos, podemos tentar ir mais longe e criar uma linguagem que sintetize os procedimentos meta-heurísticos. Como forma de resumir as meta-heurísticas descritas ao longo deste capítulo iremos criar um linguagem composta pelos operadores já definidos e por um conjunto de operandos. Como operandos heurísticos temos: alteração da vizinhança (AV), heurística construtiva (HC), heurística melhorativa (HM), multi-partida (MP) e o operador de combinação (OC). Podemos agora, redefinir as metaheurísticas como combinações de estratégias, da seguinte forma:

i) O Simulated Annealing (SA) vai restringido o espaço de vizinhança ao longo das iterações utilizando uma heurística melhorativa: SA=*(AV+HM).

ii) A Procura Tabu (PT) utiliza estratégias combinadas, no caso da alternância entre as estratégias de intensificação (HM1) e diversificação (HM2), temos: PT=*(HM1+HM2). Se utilizarmos os operandos alteração de vizinhança para intensificação (AV1) e alteração de vizinhança para diversificação (AV2), temos: PT=*(AV1+HM1+AV2+HM2)

iii) O GRASP é sem qualquer dúvida um algoritmo híbrido, com base numa nova partida executa uma heurística construtiva seguindo-se uma heurística melhorativa: GRASP= *(MP+HC+HM)

iv) O Algoritmo das Formigas (AF) repete processos construtivos orientados por alterações de vizinhança produzidos pelo feromona, temos assim: AF=*(AV+HC)

v) O Scatter Search (SS), depois de combinar soluções executa um procedimento para reparação e melhoria das soluções de teste: SS=*(OC+HM)

3.5.3- Equipas Assíncronas

O exemplo muito interessante de um algoritmo híbrido composto por um conjunto de heurísticas são as Equipas Assíncronas (Equipas-A) [Talukdar et al.1998].

Estruturalmente as Equipas-A são uma rede de dados e processos ou, mais especificamente, de memórias e de agentes autónomos. Cada memória ou estrutura de dados, representa um problema ou um sub-problema. Cada memória contem um conjunto de soluções de teste ou, por outras palavras, populações de soluções criadas pelos agentes. A existência de várias memórias significa que existem várias representações completas ou parciais do problema. Por outro lado, os agentes cooperam na criação de novas soluções de teste ou na destruição de soluções menos interessantes das populações. Os agentes incluem várias formas de resolver problemas baseando-se tanto em processos automáticos como em processos manuais.

As redes das Equipas-A podem ser vistas como hiper-grafos orientados, também chamados de fluxos de dados. O modelo para o desenvolvimento de Equipas-A pode-se apresentar em seis passos:

Procedimento das Equipas Assíncronas

1. definir o problema;
2. definir uma ou mais memórias ou sub-problemas;
3. escolher o conjunto de algoritmos construtores que actuam sobre os sub-problemas;
4. definir os agentes como sistemas de controlo, associando as memórias aos algoritmos;
5. escolher os agentes demolidores;
6. escolher o desenho do conjunto dos arcos, por forma a criar uma rede fortemente cíclica;

A rede fortemente cíclica permite uma resolução rápida do problema de optimização, facilitada pelo recurso de computadores em paralelo. As Equipas-A fornecem um modelo onde os recursos humanos e os vários tipos de agentes autónomos cooperam de forma eficiente. Uma grande variedade de soluções completas e parciais, em memórias diferentes, e um conjunto de agentes estrategicamente coordenados, permitem que as Equipas-A convirjam para soluções de alta qualidade.

As Equipas-A estendem o conceito já implementado nos Algoritmos Genéticos Distribuídos, com populações divididas em ilhas, onde a troca de indivíduos entre ilhas é reduzida e onde cada ilha se comporta como um AG isolado.

3.6 - Conclusões

Foi desenvolvida uma taxonomia para as meta-heurísticas, as ME intensivas que trabalham dentro de uma determinada região (ou intra-regionais) e as ME extensivas de varrem várias regiões (ou inter-regionais). No final do capítulo, é abordada a forma híbrida como as ME combinam as duas características, evitando as formas mais "puras" e menos eficientes. Em geral as formas híbridas dos algoritmos têm uma eficácia mais elevada do que qualquer meta-heurística no seu estado mais puro, sejam elas intensivas ou extensivas. As ME intensivas sem capacidade de mudanças de região perdem eficácia, bem como as ME extensivas que sem capacidade de aprofundar a pesquisa em determinadas regiões, não passam de algoritmos de vagueiam pelo espaço de soluções.

Pretende-se de uma ME híbrida que possua a capacidade de reconhecer as regiões mais prometedoras (extensivo) e que saiba explorar uma região específica de forma eficiente (intensiva). O processo é idêntico na pesquisa em árvore: o algoritmo é mais eficiente se souber reconhecer os melhores ramos e a seguir souber explorá-los em profundidade.

Para além da dicotomia Intensiva-Extensiva, as ME híbridas podem ser criadas conjugando duas quaisquer estratégias opostas: Primal-Dual, Construtiva-Destrutiva.

No capítulo 4 onde se desenvolve uma estratégia Primal-Dual e no capítulo 5 onde se propõem uma estratégia Intensiva-Extensiva iremos demonstrar experimentalmente que as soluções híbridas têm comportamentos mais eficazes do que as ME no seu estado mais "puro".

Bibliografia

Aggarwal C.C., Orlin J.B. e Tai R.P. (1997), "Optimized crossover for the independent set problem", *Operations Research*, vol 45, pp 226-234.

Agrawal R., Mannila H., Srikant R., Toivonen H. e Verkamo A. (1996), "Fast discovery of association rules", in *Advances in Knowledge and Data Mining*, (Fayyad U.M., Piatetsky-Shapiro G., Smyth P. e Uthurusamy R. Editors), MIT Press.

Bäck T. (1994), "Selective pressure in evolutionary algorithms: a characterization of selection mechanisms" in *Proceedings of the 1994 IEEE Conference on Evolutionary Computation* , IEEE Press, Piscataway, NJ.

Balas E., Niehaus W. (1998), "Optimized crossover-based genetic algorithms will be the maximum cardinality and maximum weight clique problems", *Journal of Heuristics,* Kluwer Academic Publishers, vol 4, pp 107-122.

Batel Anjo A., (1999) "A têmpera controlada pela entropia - uma caracterização formal", Tese de Doutoramento, Universidade de Aveiro.

Battiti R., Protasi M. (1995), "Reactive Local Search for the Maximum Clique Problem", Technical Report 75-052, International Computer Science Institute, Berkeley, CA.

Berge C. (1958), "Theórie des graphes et ses applications", Dunod, Paris.

Berge C. (1973), "Graphs and hypergraphes", Dunod, Paris.

Berge C. (1991), "Graphs", 3rd edition, North-Holland.

Berretta B. e Moscato P. (1999), "The number partitioning problem: an open challenge for evolutionary computation ?" in *New Ideas in Optimization*, (Corne D., Dorigo M., Glover F. Editors), McGraw-Hill International.

Berry M. e Linoff G. (1997), "Data Mining Techniques, for Marketing, Sales and Customer Support", John Wiley and Sons.

Brockington M. e Culberson J.C. (1996), "Camouflaging independent sets in quasi-random graphs" in *Clique, Coloring and Satisfiability*, Second Implementation Challenge DIMACS, (Johnson D.S., Trick M.A. Editors), AMS, pp 75-89.

Bron C. e Kerbosh J. (1973), "A 457- Finding all cliques of an undirected graph", *Communications of ACM*, vol 6, nº 9.

Cao B. e Glover F. (1997),"Tabu search and ejection chains - application to a node weighted version of the cardinality-constrained TSP", *Management Science*, vol 43, pp 908-921.

Cardoso M., Themido I., Moura Pires F. (1999), "Evaluating a clustering solution: an application in the tourism market", *Intelligent Data Analysis*, vol 3, pp 491-510.

Carvalho J.M.C. e Dias E.B. (2000), "e-Logistics e e-Business", Edições Sílabo.

Cavique L. (1994), "Sequenciamento de pessoal tripulante", Tese de Mestrado em Investigação Operacional e Engenharia de Sistemas, Instituto Superior Técnico, Universidade Técnica de Lisboa.

Cavique L. e Themido I. (1995), "Sequenciamento de serviços de pessoal tripulante: uma abordagem baseada num conjunto de heurísticas", *Revista de Investigação Operacion*al, vol 15, pp 123-141.

Cavique L., Rego C. e Themido I. (1999), "Subgraph ejection chains and tabu search for the crew scheduling problem", *Journal of the Operational Research Society*, vol 50, pp 608-616.

Cavique L., Rego C. e Themido I. (2001 a), "Estruturas de vizinhança e algoritmos de procura local para o problema da clique máxima", aceite para publicação na *Revista de Investigação Operacional.*

Cavique L., Rego C. e Themido I. (2001 b), "A Scatter Search Algorithm for the Maximum Clique Problem" aceite para publicação in *Essays and Surveys in Metaheuristics* (Ribeiro C. e Hansen P. Editors), Kluwer Academic Publishers.

Charon I. e Hundry O. (1999), "Principles and implementations of the noisy methods", *Proceedings Third Meta-heuristic International Conference* (MIC'99), Brazil.

Christofides N. (1975), "Graph theory: an algorithm approach", Academic Press.

Colorni A., Dorigo M., Maniezzo V., (1992), "An investigation of some properties of an ant algorithm", in *Second European Conference on Parallel Problem Solving from Nature*, (Männer R. e Manderick B. Editors), Elsevier Publishing, Brussels.

Corne D., Dorigo M., Glover F. Editors (1999), "New Ideas in Optimization", McGraw-Hill International.

Feo T.A. e Resende M.G.C. (1989), "A probabilistic heuristic for a computationally difficult set covering problem", *Operations Research Letters*, vol 8, pp 67-71.

Feo T.A., Resende M.G.C. e Smith S.H. (1994), "A greedy randomized adaptive search procedure for maximum independent set", *Operations Research*, vol 42, pp 860-878.

Ferrão F. (2000), "e-Business", Colecção E-Gestão, Tecnologias de Informação e Comunicação, Escolar Editora.

Fogel L.J., Owens A.J., Walsh M.J. (1966), "Artificial intelligendse through simulated evolution", John Wiley, New York, NY.

Friden C., Hertz A., Werra D. (1989), "STABULUS: A technique for finding stable sets in large graphs with tabu search", *Computing*, vol 42, pp 35-44.

Garey M.R. e Johnson D.S. (1979), "Computers and intractability: a guide to the theory of NP-completeness", W.H. Freeman and Company, New York.

Gendreau M. (1999), "Recent Advances in Tabu Search", comunicação na Third Meta-heuristic International Conference (MIC'99), Brazil.

Gendreau M., Soriano P., Salvail L. (1993), "Solving the maximum clique problem using tabu search approach" in *Annals of Operations Research*, (Glover F., Laguna M., Taillard E., Werra D., Baltzer J.C. Editors), AG Science Publishers, vol 41, pp 385-403.

Ghamlouche I., Crianic T.G. e Gendreau M. (2001), "Path relinking and cycle-based neighbourhoods for fixed charge, capacitated, multicommodity network design", in *Proceedings 4th Meta-heuristic International Conference (*MIC 2001), Porto.

Glover F. (1977), "Heuristics for integer programming using surrogate constraints", *Decision Science*, vol 8, pp 156-166.

Glover F. (1986), "Future paths for integer programming and links to artificial intelligence", *Computers and Operations Research*, vol 13, pp 533-549.

Glover F. (1989), "Tabu search - part I", *ORSA Journal of Computing*, vol 1, pp 190-206.

Glover F. (1997), "A template for scatter search and path relinking", in *Lecture Notes in Computer Science*, (Hao J.-K., Lutton E., Ronald E., Schoenaurer M., Snyers D. Editors) pp 1-45.

Glover F. (1999), "Scatter search and path relinking", in *New Ideas in Optimization*, (Corne D., Dorigo M., Glover F. Editors), McGraw-Hill International.

Glover F. e Laguna M. (1997), "Tabu Search", Kluwer Academic Publishers.

Goldberg D.E. (1989), "Genetic Algorithms in Search, optimization and machine learning", Addison-Wesley, Reading, MA.

Gomes C., Selman B., Crato N. (1997), "Heavy-Tailed Distributions in Combinatorial Search", in *Proc. Constraint Programming*, Linz, Austria.

Hansen P. e Mladenovic N. (1998), "An introduction to Variable Neighborhood Search", *Proceedings of the 2nd International Conference on Meta-heuristics* (MIC'97).

Holland J.H. (1975), "Adaptation in natural and artificial systems", University of Michigan Press, Ann Arbor, MI.

Hughes A.M. (2000), "Strategic Database Marketing", McGraw-Hill.

Jagota A., Sanchis L., Ganesan R. (1996), "Approximately Solving Maximum Clique using Neural Network Related Heuristics", in *Clique, Coloring and Satisfiability*, Second Implementation Challenge DIMACS, (Johnson D.S., Trick M.A. Editors), AMS, pp 169-203.

Johnson D.S. (1974), "Approximation algorithms for combinatorial problems", *Journal of Computer and System Sciences*, vol 9, pp 256-278.

Johnson D.S. e Trick M.A. eds (1996), "Clique, Coloring and Satisfiability – Second Implementation Challenge DIMACS", AMS.

Kelly J.P., Golden P.L., Assad A.A. (1993), "Large-scale controlled rounding using tabu search with stategic oscillation" in *Annals of Operations Research*, (Glover F., Laguna M., Taillard E., Werra D., Baltzer J.C. Editors), AG Science Publishers, vol 41, pp 69-84.

Kirkpatrick S., Gellat C.D. e Vecchi M.P. (1983), "Optimization by simulated annealing", *Science*, vol 220, pp 671-680.

Koza J.R. (1992), "Genetic programming", MIT Press, Cambridge, MA.

Laguna M. (1999), "Scatter Search", Research Report, University of Colorado, Boulder.

Laudon K. e Laudon J. (1993), "Business information systems: a problem solving approach", Dryden Press.

Lin S. (1965),"Computer solutions of the traveling salesman problem", *Bell Systems Tech. J.*, vol 44, pp 2245-2269.

Lourenço H., Paixão J. e Portugal R. (1999), "Meta-heuristics for the bus-driver scheduling problem", Economic Working Papers Series 304, Universitat Pompeu Fabra, Spain.

Luz C. (1996), "Um novo majorante para o número de independência de um grafo obtido por técnicas de programação quadrática", Tese de Doutoramento, Universidade de Aveiro.

McHugh J.A. (1990), "Algorithm Graph Theory", Prentice-Hall Inc.

Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Tellet A.H e Teller E.(1953), "Equation of state calculation by fast computing machines", *J. of Chem. Phys.*, vol 21, pp 1087-1091.

Michalewicz Z. e Fogel D.B. (2000), "How to solve it: modern heuristics", Springer-Verlag.

Michalewicz Z.(1996), "Genetic Algorithms+Data Structures =Evolution Programs", Springer, Berlin.

Moscato P. (1989), "On evolution, search, optimization, genetic algorithms and marcial arts: towards memetic algorithms", Report 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, California, USA.

Peppers D. e Rogers M. (1997), “Enterprise One to One”, disponível em <URL:http://www.1to1.com>

Phillips E.M. e Pugh D.S. (1993), "How to get a PhD, a handbook for students and their supervisors", Open University Press.

Pirlot M. (1996), "General local search methods", *European Journal of Operational Research*, vol 92, pp 493-511.

Rechenberg I. (1973), "Evolutions strategie: optimierung technischer systeme nach prinzipien der biologischen evolution", Frommann-Holzboog Verlag, Stuttgart.

Reeves C.R. (1995), "Modern heuristic techniques for combinatorial problems", McGraw-Hill International.

Rego C. (1998 a), "A subpath ejection method for the vehicle routing problem", *Management Science*, vol 44, pp 1447-1459.

Rego C. (1998 b), "Relaxed tours and path ejection for the traveling salesman problem", *Eur. J. Op. Res.*, vol 106, pp 552-538.

Rego C. e Roucairol C. (1996), "Parallel tabu search algorithm based on ejection chains for the vehicle routing problem" in *Metaheuristics: theory and applications*, (Osman I.H. e Kelly J.P. Editors), Kluwer Academic Publishers, Boston, MA.

Roy B. (1970), "Algèbre moderne et théorie des graphes", tome 2, Dunod, Paris.

Salton G. e Lesk M.E. (1965), "The SMART automatic document retrieval system – an illustration", *Communications of the ACM*, vol 8, nº 6, pp 391-398.

Salton G. e McGill M.J. (1983), "Introduction to Modern Information Retrieval", McGraw-Hill.

SAS Software (2000), "Enterprise Miner Documentation", SAS Institute

Schwefel H.-P. (1981), "Numerical optimization for computer models", John Wiley, Chichester, UK.

Soriano P., Gendreau M. (1996), "Tabu search algorithms for the maximum clique", in *Clique, Coloring and Satisfiability*, Second Implementation Challenge DIMACS, (Johnson D.S., Trick M.A. Editors), AMS, pp 221-242.

Taillard É.D., Gambardella L.-M., Gendreau M., Potvin J.-Y. (1998), "Adaptive Memory Programming: A Unified View of Meta-Heuristics", Technical report IDSIA-19-98, IDSIA, Lugano, publicado no *EURO XVI Conference Tutorial and Research Reviews*, Brussels.

Talukdar S., Baerentzen L., Gove A. e Souza P. (1998), "Asynchronous Teams: Cooperation Schemes for Autonomous Agents", *Journal of Heuristics*, vol 4, pp 295-321.