Métodos Modernos de Pesquisa Heurística

1 - Introdução aos conceitos.

A classe de algoritmos que aqui se considera pode também ser designada por "*Algoritmos Iterativos de Melhoramento*" (Iterative Improvement Algorithms). Mas melhoramento de quê? e em que sentido?. Estes algoritmos partem de uma ou mais soluções potenciais do problema e, dispondo de uma medida da sua *qualidade*, tentam fazer alterações a essa(s) solução(ões) de modo a melhorar a sua qualidade por forma a que, ao fim de um certo número de iterações, a qualidade da(s) solução(ões) seja óptima. Trata-se portanto de formular o problema como um problema de optimização, sendo para tal necessário definir, para cada problema concreto: qual o espaço de potenciais soluções e como são descritas, i.e. qual a sua representação *x*; como se mede a qualidade das potenciais soluções, ou seja qual a função de optimização **f**(**x**), e o que se entende, para o problema específico, por "pequenas alterações", ou seja, qual a vizinhança de uma um potencial solução ***V*(*x*)**.

Estes métodos não garantem, em geral, que um óptimo global seja encontrado, podendo ficar retidos em óptimos locais. No entanto, podem em muitos casos obter rapidamente soluções de boa qualidade, mesmo em problemas de grandes dimensões (dimensão do espaço de pesquisa).

Para além da escolha acertada dos valores dos parâmetros, específicos de cada algoritmo, a escolha da representação das soluções potenciais, da vizinhança e da função a optimizar, constituem os aspectos centrais da aplicação destes algoritmos a problemas concretos.

2 - Espaço de soluções potenciais, vizinhança e função de optimização

Para cada problema é necessário proceder à sua formulação como um problema de optimização. Para tal, é necessário determinar qual o espaço de potenciais soluções, qual a estrutura e dimensão da vizinhança proposta e a função a optimizar. Analisemos alguns destes aspectos para o caso do problema do caixeiro viajante:

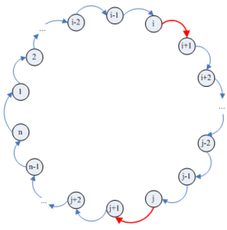
Se temos *n* cidades, qualquer volta pode ser representada como uma permutação dos números de 1 a *n*. Isto corresponde fisicamente a um circuito ligando as cidades pela ordem indicada pela permutação e considerando que existe uma ligação entre a última cidade e a primeira. Por exemplo a permutação [3, 2, 1, 4, 5] corresponde a um circuito que liga as cidades 3 -> 2 -> 1 -> 4 -> 5 -> 3.

O conjunto destas permutações constitui o espaço das potenciais soluções, cuja cardinalidade é *n*!. Repare-se no entanto que uma permutação circular de qualquer sequência de números representa o mesmo circuito, pelo que podemos fixar qualquer cidade como a de partida resultando assim num decréscimo no tamanho do espaço de soluções para (*n*-1)! Além disso, deve ser observado que se a distância entre duas cidades for independente do sentido - distância simétrica - o número de circuitos diferentes é então de (*n*-1)!/2.

Uma vizinhança-K de um circuito é definida pelo conjunto de todos os circuitos que se obtêm removendo *K* ligações e substituindo-as por *K* outras ligações, de tal modo que se mantenha um circuito válido. Para K > 3 existem diferentes maneiras de, depois de removidas as *K* ligações, se voltar a ligar os circuito. Para K = 2, existe apenas uma maneira de voltar a fazer as ligações: se se retirarem as ligações *(i, i+1)* e *(j, j+1) (com j>i+1, ou seja com i e j distintos e não consecutivos)* então a única maneira admissível de criar um outro circuito é ligando *(i, j)* e *(i+1, j+1)*, e invertendo a direcção das ligações entre *i*+1 e *j* (***vizinho***).

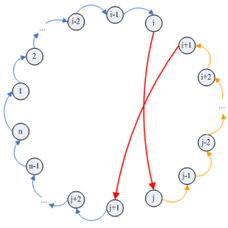
Considere por exemplo o circuito que se mostra na figura seguinte, que corresponde à representação de

[1, 2, ..., *i*-2, *i*-1, *i*, *i*+1, *i*+2, ..., *j*-2, *j*-1, *j*, *j*+1, *j*+2, ..., *n*-1, *n*].



Se as ligações nas posições i e j, ou seja *(i, i+1)* e *(j, j+1) (com j>i+1)*, forem removidas, então as únicas ligações que podemos acrescentar, de modo a obter um circuito, são *(i, j)* e *(i+1, j+1)*, e invertendo a direcção das ligações entre *i*+1 e *j*. O novo circuito, assim obtido, tal como se mostra na figura abaixo, é representado por

[1, 2, ..., *i*-2, *i*-1, *i*, *j*, *j*-1, *j*-2, ..., *i*+2, *i*+1, *j*+1, *j*+2, ..., *n*-1, *n*].



Deste modo, cada par de valores *i* e *j (com j>i+1)*, representa um vizinho. A dimensão da vizinhança é limitada por *n*(*n*-1)/2, o número de combinações de *n* dois a dois.

Qualquer circuito pode ser obtido pela aplicação de uma ou mais vezes (número finito) desta operação a partir de outro circuito qualquer. Por outras palavras, qualquer ponto do espaço é atingível a partir de qualquer outro ponto do espaço pela aplicação sucessiva do operador de vizinhança.

Adicionalmente, esta estrutura de vizinhança é fácil de ser implementada pois consiste apenas na troca de valores entre duas posições, posições essas que podem ser escolhidas aleatoriamente extraindo sem reposição dois números *i* e *j*, não consecutivos, entre 1 e *n*.

A função a optimizar é naturalmente a soma das distâncias. No entanto, o que os algoritmos necessitam em geral é conhecer apenas qual foi a variação na qualidade da solução. Representando por *d* essa variação, calcula-se:

$$d = d(i,j) + d(i+1,j+1) - d(i,i+1) - d(j,j+1)$$

Finalmente, o ponto inicial pode ser obtido facilmente (***cria_solução_inicial***), pois é fácil gerar aleatoriamente uma solução aceitável: biragem sem reposição de *n* números de 1 a *n*.

Depois de escolhida a forma de modelação e a vizinhança, é conveniente ter uma ideia acerca de:

1. Qual a dimensão do espaço de soluções potenciais;
2. Qual a dimensão da vizinhança;
3. Qual é a máxima variação da função de custo entre quaisquer dois vizinhos. Se não for possível determinar este valor a partir do problema, é possível obter uma estimativa gerando aleatoriamente (uniformemente) um conjunto de pontos do espaço de potenciais soluções e calculando os respectivos valores da função de custo. Podemos assim ter uma ideia da função para a qual se irá procurar uma solução óptima, bem como uma estimativa da variação máxima da função de custo.

3 - Simulated Annealing

A aplicação do SA envolve a sua adaptação ao problema concreto, nomeadamente:

- Função de probabilidade de aceitação, em geral na forma P(*d*) = exp(-*d*/T)
- Temperatura inicial, T_0 (***temperatura_inicial***)
- Número de iterações à mesma temperatura (***n_iter***, ***var_n_iter***)
- Função de decaimento da temperatura, em geral na forma $T_k = f(T_{k-1})$ (***decaimento***)
- Condição de paragem (***criterio_de_paragem***).

Embora não seja possível um método geral para estas decisões, existem algumas recomendações úteis:

1. Determinação da temperatura inicial (***temperatura_inicial***)

   Se pretendemos que a solução final seja o mais independente possível do ponto inicial, então a temperatura inicial deve ser suficientemente alta para que qualquer vizinho seja aceite.

   - Em alguns casos é possível estimar um valor de T_0 para esse efeito. Se o d_{max} entre quaisquer dois vizinhos for conhecido, então podemos regular o valor de T_0 para que, para esse valor de d_{max}, a probabilidade de aceitação seja grande. Por exemplo, no caso do caixeiro viajante, quando um vizinho se obtém eliminando duas ligações e substituindo-as por duas outras, o d_{max} ocorre quando se retiram as duas menores distancias e se juntam as duas maiores distancias.
   - Quando não se dispõe de tal informação, é frequente o sistema iniciar-se com uma temperatura não muito alta, e ir subindo a temperatura (em geral rapidamente) até que a proporção de vizinhos aceites seja suficientemente elevada, e só então se inicia o processo de arrefecimento.
2. Procedimento de arrefecimento (***decaimento***, ***n_iter***, ***var_n_iter***)

   Este depende do número de repetições em cada temperatura e do ritmo de decaimento da temperatura. Dois esquemas de arrefecimento são frequentes na literatura que representam extremos opostos (naturalmente existem outras abordagens):

   - Redução geométrica: $T_k = \alpha\ T_{k-1}$, com $\alpha < 1$. Na prática, valores altos de α têm-se revelado eficazes (entre 0.8 e 0.99). Quanto mais alto o valor de α mais lento é o arrefecimento.
     O número de iterações em cada temperatura está em geral relacionado com a dimensão da vizinhança, ou por vezes com a dimensão do espaço de soluções. Além disso, este número (**n-iter**) pode variar com a temperatura (maior número de iterações a baixas temperaturas). Por exemplo, de cada vez que se muda de temperatura multiplica-se n-iter por uma constante um pouco maior que 1 (variação geométrica de n-iter), ou adiciona-se um certo número fixo de iterações (variação aritmética de n-iter).
     Outra forma de controlar o n-iter é impor um número máximo de iterações por temperatura, mas autorizar a mudança de temperatura desde que um certo número de vizinhos tenham sido aceites.
   - Outro procedimento de arrefecimento consiste em executar apenas uma iteração a cada temperatura mas reduzir a temperatura de forma muito gradual de acordo com a formula: $T_k = T_{k-1} / (1 + \beta\ T_{k-1})$ onde β é um número adequadamente pequeno.
3. Condição de paragem (***criterio_de_paragem***)

   Teoricamente a temperatura deveria descer até zero antes de ocorrer a condição de paragem. No entanto não há, na prática, necessidade disso, até devido à limitada precisão dos computadores. Na prática a condição de paragem pode ser expressa não directamente mas em termos da "rigidez" do sistema, isto é, em termos da proporção de movimentos aceites.

   O processo pára quando a proporção de movimentos aceites desce abaixo de um certo nível. Uma forma ainda mais simples de implementar, mas mais dependente do problema (em particular da sua dimensão, e os outros parâmetros), consiste na paragem ao fim de um certo número de iterações.

4. Guardar o melhor de sempre

   A altas temperaturas, a probabilidade de o SA sair de pontos óptimos e ir para pontos de menor qualidade não é negligenciável, pelo que o ponto encontrado não é necessariamente o melhor de todos os pontos visitados. Assim, é conveniente guardar um registo do melhor ponto encontrado durante a pesquisa.