UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

RELATÓRIO DO TRABALHO DA DISCIPLINA
ACH2016 - Inteligência Artificial

Integrantes:	NºUSP:
André Ramos	9125339
Matheus Mendes de Sant'Ana	8921666
Victor Luiz Soares Alexandre Pereira	8921350

SÃO PAULO

2016

1-) Resumo:

Este relatório visa descrever e detalhar todo o processo da elaboração de um algoritmo para solucionar o Problema de Roteamento de Veículos (Vehicle Routing Problem – VRP), que será explicado mais adiante. Para isso, será implementado em java o método meta-heurístico Simulated Annealing, onde apresentaremos o seu pseudocódigo e uma descrição de como funciona o algoritmo em si. Após isso, serão mostrados o diagrama de classes do algoritmo, e as dificuldades encontradas durante toda a implementação da solução.

Serão expostas as configurações utilizadas nos experimentos, e ainda, uma tabela com os resultados obtidos. Por fim, faremos uma análise geral dos resultados e uma breve conclusão de todo o trabalho desenvolvido.

2-) Introdução:

O problema de roteamento de veículos (VRP - *Vehicle Routing Problem*) é uma otimização de um problema de programação combinatória e de programação com números inteiros. O algoritmo busca atender um número de clientes com uma frota limitada de veículos, com base em um ou vários depósitos que devem ser determinados por uma série de cidades ou clientes geograficamente dispersos, com o objetivo de entregar produtos a todo esse conjunto de clientes com demandas conhecidas em rotas de veículos de custo mínimo com origem e destino em um depósito. Proposto por George Dantzig e John Ramser em 1959, VRP é um problema importante nas áreas de transporte, distribuição, logística e socioambiental, pois um roteamento bem escolhido implica em menos emissão de poluentes, maior eficiência na execução do trabalho e, consequentemente, menor gasto para a empresa.

3-) Definição do Problema VRP Capacitado:

Neste trabalho será utilizado uma variação do VRP, o CVRP – Capacitated Vehicle Routing Problem, ou seja, Problema de Roteamento de Veículos Capacitado, em que existe uma restrição que cada veículo possui uma capacidade de carga limitada e uniforme para as mercadorias. O objetivo do CVRP é minimizar o tamanho da frota de veículos usados, e diminuir a soma do tempo de viagem, lembrando que a demanda total de mercadorias para cada rota não pode exceder a capacidade do veículo que serve esse caminho.

4-) Pseudocódigo e descrição do algoritmo escolhido:

A seguir será apresentado o pseudocódigo escolhido para o desenvolvimento do algoritmo que será utilizado na resolução do CVRP:

```
SimulatedAnnealing() {
    Lista sequenciaEstados;
    for(vertice : Grafo) {
        if(vertice != depot) {
            sequenciaEstados.adicionar(vertice);
        }
    }
    tamanhoLista = tamanho(sequenciaEstados);

    temperatura = tamanhoLista;
    taxaResfriamento = 1/(tamanhoLista)²;
    limiteTemperatura = taxaResfriamento;
    ultimaModificacao = (tamanhoLista)²;

    distanciaTotalAnterior = -1;
    distanciaTotalAtual = -1;
    caminhoesUsadosAnterior = -1;
    caminhoesUsadosAtual = -1;

    dividirEmRotasDosCaminhoes(sequenciaEstados);
    distanciaTotalAnterior = distanciaTotalConfiguraçaoAtual();
    caminhoesUsadosAnterior = caminhoesUsadosConfiguraçaoAtual();

    while(temperatura >= limiteTemperatura || ultimaModificacao > 0) {
        posicaoSwap1 = RandomPosicao(sequenciaEstados);
        posicaoSwap2 = RandomPosicao(sequenciaEstados);
        while(posicaoSwap1==posicaoSwap2) {
            posicaoSwap1 = RandomPosicao(sequenciaEstados);
            posicaoSwap2 = RandomPosicao(sequenciaEstados);
        }
        fazerSwap(posicaoSwap1, posicaoSwap2, sequenciaEstados);
        distanciaTotalAtual = distanciaTotalConfiguraçaoAtual();
```

```
                caminhoesUsadosAtual = caminhoesUsadosConfiguraçaoAtual();
                boolean modificou = false;

                if (distanciaTotalAtual > distanciaTotalAnterior) {
                        deltaE = distanciaTotalAnterior – distanciaTotalAtual;
                        if(!aceita(deltaE, temperatura)) {
                                voltarSolucaoComoEraAntes();
                        }
                        else {
                                distanciaTotalAnterior = distanciaTotalAtual;
                                caminhoesUsadosAnterior = caminhoesUsadosAtual;
                                modificou = true;
                        }
                }
                else if(distanciaTotalAnterior==distanciaTotalAtual) {
                        if(caminhoesUsadosAtual<=caminhoesUsadosAnterior) {
                                distanciaTotalAnterior = distanciaTotalAtual;
                                caminhoesUsadosAnterior = caminhoesUsadosAtual;
                                modificou = true;
                        }
                        else {
                                voltarSolucaoComoEraAntes();
                        }
                }
                else { //Se solução atual é melhor que solução anterior
                        distanciaTotalAnterior = distanciaTotalAtual;
                        caminhoesUsadosAnterior = caminhoesUsadosAtual;
                        modificou = true;
                }
                if(modificou == true) ultimaModificacao += 1;
                else ultimaModificacao -=1;
                temperatura = temperatura - (temperatura*taxaResfriamento);
        }
}
```

Simulated Annealing (Recozimento Simulado) é um algoritmo meta-heurístico de otimização com busca local probabilística proposto por Scoot Kirkpatrick em 1983, fazendo uma analogia ao processo de recozimento de metais, na área da termodinâmica. O programa funciona da seguinte maneira: os estados possíveis de um metal correspondem às soluções do espaço de busca. A energia em cada estado equivale ao valor da função objetivo. A energia mínima, ou máxima dependendo do problema, corresponde ao valor de uma solução ótima local, possivelmente global.

A cada iteração do método, um novo estado é gerado a partir do estado corrente por uma modificação aleatória, e se o novo estado é de energia melhor que o estado corrente, esse estado passa a ser o estado corrente, caso seja pior é avaliado a probabilidade de se mudar ou não de estado. Todo esse procedimento é repetido até se atingir o melhor resultado.

Em sua base, o algoritmo desenvolvido no trabalho faz uso dos mesmos princípios do Simulated Annealing, com a definição de uma temperatura inicial, de uma taxa de resfriamento e da perturbação do meio enquanto o critério de parada não for satisfeito. A utilização de uma função aleatória que aceita ou não uma piora na solução também foi implementada, possui o nome “aceita()”, e pode ser encontrada na classe “SimulatedAnnealing”. A função “aceita()” utiliza a classe “Random” que, por sua vez, faz uso da classe “Math” original do Java para gerar números aleatórios de forma mais randômica possível.

Uma solução é gerada através de uma sequência de estados em uma determinada lista (no caso, utilizamos a lista “sequenciaEstados”), onde deve conter todos os vértices do grafo exceto o vértice “depot”; tendo a lista cumprido esse requisito, chamamos a função “transferirParaSolucao()” da classe “SimulatedAnnealing” tendo como parâmetro a lista definida com o requisito. Desta forma, a função “transferirParaSolucao()” desmembra a lista na ordem exata que os vértices foram inseridos, dividindo por rotas de caminhão cada conjunto de vértice que não exceda a capacidade dos caminhões.

Dado o método descrito anteriormente que transfere uma sequência dos estados para a solução, uma das responsabilidades restantes do algoritmo é perturbar o meio (no caso a que contém todos os vértices menos “depot”) até que se encontre um estado satisfatório: o “swap”, ou troca de posições de vértices, é utilizado para perturbar o meio em questão.

O critério do mínimo de veículos utilizados também teve de ser implementado no algoritmo (não existente originalmente no Simulated Annealing), que consiste em verificar a melhoria de uma solução durante o laço de repetição: se a solução anterior e a solução atual possuem o mesmo custo de distância total, a solução atual (lista “solucao” da classe “Caminhos”) receberá aquela que utiliza menos veículos; se ambas utilizarem a mesma quantidade de veículos a solução atual continuará presente na lista de soluções (da classe “Caminhos”), pois o custo de se rearranjar a solução atual traria ineficiência (realizar novamente “swap” para manter estado anterior gastaria mais tempo). Note que não necessariamente o mínimo de veículos será obedecido pela solução final do algoritmo, pois em primeiro lugar o algoritmo analisa o menor custo total das rotas, e em segundo lugar ele analisa o menor número de caminhões utilizado. Todavia, na maior parte dos casos a quantidade mínima de veículos é apresentada na solução.

Critério de Parada:

O principal desafio do algoritmo foi encontrar um critério de parada que ao mesmo tempo favorecesse o encontro de uma **boa solução final** e um **gasto aceitável de tempo de espera**. Como será dado em seguida, as variáveis que controlam esses critérios obedecem o conceito de que para um problema com mais vértices, o gasto de tempo médio será maior do que um problema com menos vértices (decorrente de mais comparações possíveis permitidas ao problema com mais vértices). Para satisfazermos todas essas condições, propusemos os seguintes critérios de parada:

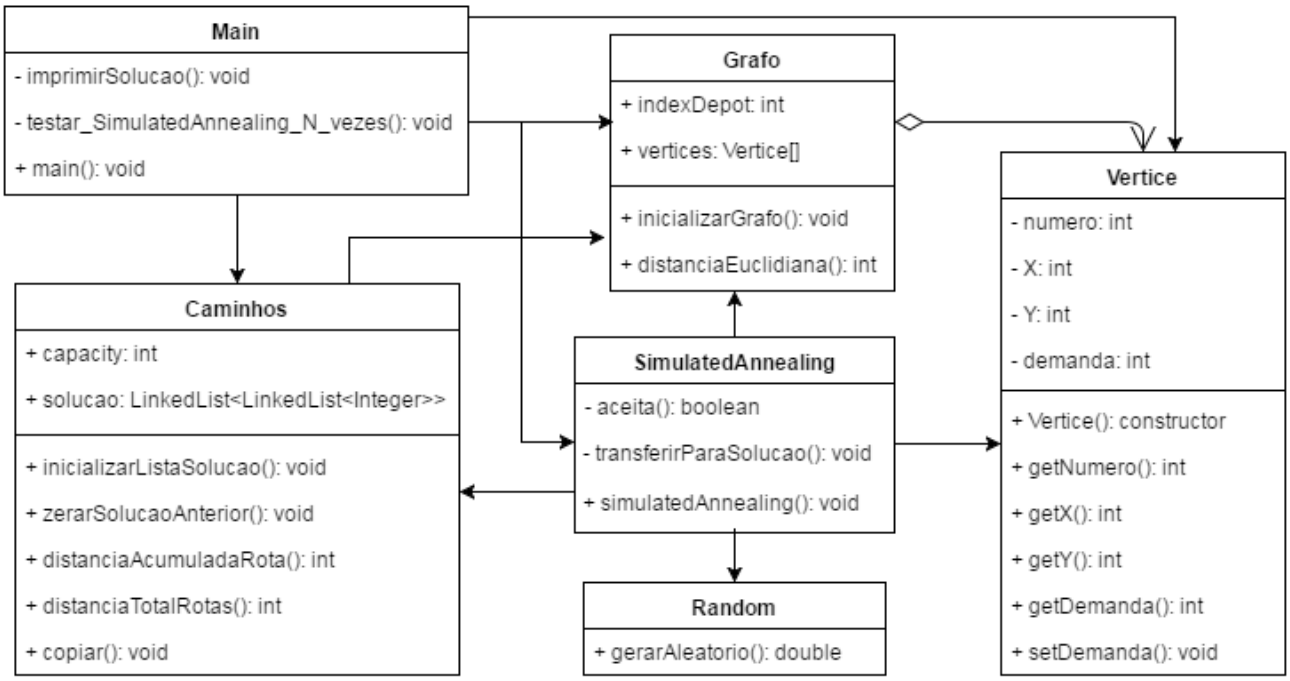
1- Temperatura, taxa de resfriamento e limite da temperatura (inicialmente proposto pelo algoritmo Simulated Annealing):

Para cada entrada de um problema, a temperatura é inicializada com tamanho n, onde n é igual ao número de vértices -1 (retirando "depot" da comparação). A escolha do tamanho inicial da temperatura se deve ao conceito de átomos do próprio algoritmo Simulated Annealing, dado que o número de átomos seria o número de vértices que pretendemos comparar, e a temperatura receberia a ordem de grandeza da quantidade desses átomos (vértices) = O(n). Já para a taxa de resfriamento, o primeiro aspecto seria definir um número maior que 0 e menor que 1, de onde veio o valor $1/n^2$, que seria o inverso da ordem de grandeza das comparações entre os n vértices, dada como $O(n^2)$. Como não é possível que no programa a temperatura chegue a zero, mas chegue muito próximo disso (conceito de limite), foi definido o limite de temperatura como sendo a própria taxa de resfriamento, dado que ela é próxima de zero.

2- Última modificação:

permite que o algoritmo continue executando enquanto o número de modificações possíveis for maior que zero. Inicialmente, a variável "ultimaModificacao" que controla o número de modificações é definida com tamanho n^2, onde n é igual ao número de vértices do grafo -1 (menos o vértice "depot"). A escolha do tamanho inicial desta variável foi decorrente de uma suposição sobre a ordem de grandeza do número de comparações entre vértices que não sejam o "depot", de onde parte o tamanho n^2: se o número possível de laços repetidos sem modificação foi alcançado, esse segundo critério de parada já foi atingido. Para uma modificação efetuada, a variável é incrementada; caso contrário, ela é decrementada. Foi utilizada uma heurística para a criação desse critério de parada: para uma solução ótima, serão executadas pelo menos $(n+1)^2 = O(n^2)$ comparações (onde n soma-se ao "depot" nesse caso); logo pressupõe-se que se a variável "ultimaModificacao" chegou a zero, todas as comparações necessárias entre os vértices foram efetuadas (regra "relaxada").

5-) Implementação – Diagrama de Classes:



6-) Implementação – Dificuldades encontradas durante a implementação:

Geração de números *random* da classe "Math" Java não nos forneceu valores esperados para um bom espalhamento de números aleatórios, por conta disso, foi criada a classe "Random" que visa retirar um pouco mais da previsibilidade dos números pseudoaleatórios gerados. Para isso, foi gerado um vetor desses números, depois é escolhido aleatoriamente uma posição para um número representativo dada uma chamada do método "gerarAleatorio()". Além disso, como mencionado anteriormente, a decisão de um critério de parada.

7-) Experimentos – Configuração:

No método "main" da classe "Main" foram utilizadas 10 chamadas para o algoritmo (utilizando a chamada do método "testar_SimulatedAnnealing_N_vezes()") com o intuito de encontrar melhores soluções em todos os casos de entrada. Caso seja necessário um menor gasto de tempo, menos chamadas podem ser efetuadas; caso haja tempo disponível, mais chamadas podem ser efetuadas, dada a preferência do usuário.

8-) Experimentos – Resultados e Soluções Ótimas:

A-n32-k5:

Rota#1: 0 29 18 8 11 4 28 23 3 2 26 0 custo: 257 demanda atendida: 94
Rota#2: 0 6 7 16 1 12 0 custo: 141 demanda atendida: 86
Rota#3: 0 13 17 19 31 21 30 0 custo: 155 demanda atendida: 94
Rota#4: 0 20 22 9 15 10 25 5 0 custo: 244 demanda atendida: 89
Rota#5: 0 27 24 14 0 custo: 64 demanda atendida: 47
Custo 861
Tempo 654

Solução Ótima A-n32-k5:

Route #1: 21 31 19 17 13 7 26
Route #2: 12 1 16 30
Route #3: 27 24
Route #4: 29 18 8 9 22 15 10 25 5 20
Route #5: 14 28 11 4 23 3 2 6
cost 784

A-n33-k5:

Rota#1: 0 29 3 9 17 30 25 0 custo: 203 demanda atendida: 98
Rota#2: 0 27 5 26 7 8 13 32 0 custo: 177 demanda atendida: 88
Rota#3: 0 2 11 31 18 28 23 0 custo: 91 demanda atendida: 94
Rota#4: 0 20 4 12 10 16 15 22 0 custo: 148 demanda atendida: 97
Rota#5: 0 1 21 14 19 6 24 0 custo: 114 demanda atendida: 69
Custo 733
Tempo 640

Solução Ótima A-n33-k5:

Route #1: 15 17 9 3 16 29
Route #2: 12 5 26 7 8 13 32 2
Route #3: 20 4 27 25 30 10
Route #4: 23 28 18 22
Route #5: 24 6 19 14 21 1 31 11
cost 661

A-n33-k6:

Rota#1: 0 10 19 20 9 15 2 3 5 0 custo: 174 demanda atendida: 91
Rota#2: 0 28 27 30 16 25 21 0 custo: 110 demanda atendida: 100
Rota#3: 0 1 18 6 7 13 0 custo: 75 demanda atendida: 88
Rota#4: 0 8 4 26 22 24 23 31 0 custo: 195 demanda atendida: 98
Rota#5: 0 32 11 29 17 14 0 custo: 143 demanda atendida: 98
Rota#6: 0 12 0 custo: 70 demanda atendida: 66
Custo 767
Tempo 681

Solução Ótima A-n33-k6:

Route #1: 5 2 20 15 9 3 8 4
Route #2: 31 24 23 26 22
Route #3: 17 11 29 19 7
Route #4: 10 12 21
Route #5: 28 27 30 16 25 32
Route #6: 13 6 18 1 14
cost 742

B-n31-k5:

Rota#1: 0 4 29 22 17 13 30 0 custo: 108 demanda atendida: 98
Rota#2: 0 8 23 12 28 26 0 custo: 105 demanda atendida: 78
Rota#3: 0 1 19 24 14 2 10 27 20 0 custo: 243 demanda atendida: 93
Rota#4: 0 9 6 3 11 15 7 0 custo: 150 demanda atendida: 87
Rota#5: 0 16 18 5 25 21 0 custo: 91 demanda atendida: 56
Custo 697
Tempo 622

Solução Ótima B-n31-k5:

Route #1: 30 23 8 12 28 26
Route #2: 21 16 18 25 5 4 29
Route #3: 7 17 13 6 9 22
Route #4: 20 27 10 2
Route #5: 14 15 11 24 19 1 3
cost 672

B-n34-k5:

Rota#1: 0 11 30 7 21 32 8 4 0 custo: 215 demanda atendida: 100
Rota#2: 0 1 19 18 16 14 24 29 27 0 custo: 153 demanda atendida: 90
Rota#3: 0 2 22 33 10 23 20 0 custo: 157 demanda atendida: 99
Rota#4: 0 5 25 17 31 28 13 15 6 0 custo: 112 demanda atendida: 86
Rota#5: 0 9 3 26 12 0 custo: 162 demanda atendida: 82
Custo 799
Tempo 670

Solução Ótima B-n34-k5:

Route #1: 33 32 22 30 11
Route #2: 15 13 28 31 17 25 6 5
Route #3: 14 9
Route #4: 12 26 3 27 29 24 16 18 19 1
Route #5: 8 10 23 20 4 21 7 2
cost 788

B-n35-k5:

Rota#1: 0 20 7 12 16 10 17 29 0 custo: 113 demanda atendida: 91
Rota#2: 0 22 5 11 19 15 2 32 0 custo: 231 demanda atendida: 100
Rota#3: 0 34 27 14 24 4 28 30 13 0 custo: 268 demanda atendida: 97
Rota#4: 0 9 23 1 6 33 3 18 26 0 custo: 245 demanda atendida: 93
Rota#5: 0 8 31 25 21 0 custo: 131 demanda atendida: 56
Custo 988
Tempo 740

Solução Ótima B-n35-k5:

Route #1: 34 27 24 28 4 14 22
Route #2: 29 17 10 16 12
Route #3: 21 25 31 8
Route #4: 15 30 5 11 19 2 32 13
Route #5: 20 7 9 23 1 26 18 3 33 6
cost 955

P-n16-k8:

Rota#1: 0 1 3 0 custo: 66 demanda atendida: 35
Rota#2: 0 8 0 custo: 64 demanda atendida: 28
Rota#3: 0 4 11 0 custo: 57 demanda atendida: 30
Rota#4: 0 7 9 13 0 custo: 68 demanda atendida: 29
Rota#5: 0 2 0 custo: 42 demanda atendida: 30
Rota#6: 0 10 12 15 0 custo: 67 demanda atendida: 33
Rota#7: 0 6 0 custo: 24 demanda atendida: 31
Rota#8: 0 5 14 0 custo: 62 demanda atendida: 30
Custo 450
Tempo 144

Melhor Custo Encontrado P-n16-k8 (Visualizado no próprio arquivo): 435

Maior Custo/ Menor Custo:

Solução do algoritmo/ Melhor solução encontrada = 450/435 = 1,0344

Conclusão: 3,44% de diferença entre a melhor solução encontrada e a solução do algoritmo.

P-n19-k2:

Rota#1: 0 18 5 13 15 9 7 2 10 1 0 custo: 98 demanda atendida: 157
Rota#2: 0 4 11 14 12 3 17 16 8 6 0 custo: 114 demanda atendida: 153
Custo 212
Tempo 170

Melhor Custo Encontrado P-n19-k2 (Visualizado no próprio arquivo): 212

Maior Custo/ Menor Custo:

Solução do algoritmo/ Melhor solução encontrada = 212/212 = 1

Conclusão: Melhor solução encontrada e a solução do algoritmo possuem o mesmo custo: excelente desempenho do algoritmo neste problema.

P-n20-k2:

Rota#1: 0 19 5 14 16 9 7 2 6 0 custo: 89 demanda atendida: 155
Rota#2: 0 4 11 15 12 3 18 17 8 13 10 1 0 custo: 128 demanda atendida: 155
Custo 217
Tempo 180

Melhor Custo Encontrado P-n20-k2 (Visualizado no próprio arquivo): 220

Maior Custo/ Menor Custo:

Melhor solução encontrada/ Solução do algoritmo = 220/217 = 1,0138

Conclusão: Solução do algoritmo **superou** a melhor solução encontrada: excelente desempenho do algoritmo neste problema, redefinindo o conceito de melhor solução encontrada. (1,38% de diferença entre a melhor solução encontrada e a solução do algoritmo.)

9-) Experimentos – Análise Geral:

A seguir faremos uma tabela comparativa entre os valores de obtidos e os valores ótimos de custo (apenas das entradas das classes A e B), posteriormente faremos uma análise geral dos resultados:

Testes:	Custo Obtido:	Custos Ótimos:	Comparativo:
A-n32-k5	861	784	9,82%
A-n33-k5	733	661	10,89%
A-n33-k6	767	742	3,37%
B-n31-k5	697	672	3,72%
B-n34-k5	799	788	1,40%
B-n35-k5	988	955	3,46%

Podemos perceber que os resultados obtidos são próximos, ainda que não sejam iguais aos valores ótimos, onde no pior caso os valores obtidos ficam em torno de 10% menos eficientes que o valor ótimo (em relação ao custo da solução e não ao gasto de tempo de processamento). Entretanto, em boa parte dos testes existe uma aproximação maior entre os valores. Concluímos portanto que o algoritmo elaborado neste trabalho traz resultados e gasto de tempo (muito menor que em um algoritmo ótimo) satisfatórios em seus testes comparativos. Vale ressaltar também que no método "main" utilizamos a chamada do método "testar_SimulatedAnnealing_N_vezes()" com parâmetro **10** para que houvesse um gasto de tempo em torno de no máximo 1 segundo. Caso houvesse menor rigor quanto ao tempo e a procura por uma solução mais próxima da ótima, poderia ser utilizado o teste com parâmetros maiores (como por exemplo 50).

10-) Conclusões:

A partir das análises feitas no decorrer do trabalho, podemos concluir que depois de uma breve introdução dos conceitos abordados, elaboramos, com auxílio aos ensinamentos adquiridos em sala, um algoritmo que resolve o CVRP com um bom desempenho na maior parte dos casos (gasto aceitável de tempo), trazendo resultados satisfatórios comparativamente aos valores ótimos apresentados, dado que o algoritmo não utilizou nenhum código pronto. Mais do que resultados, trouxemos uma visão detalhada de todo o funcionamento do Simulated Annealing para a resolução do CVRP, demonstrando na prática todas as vantagens e desvantagens de sua aplicação.

11-) Bibliografia:

https://www.prp.rei.unicamp.br/pibic/congressos/xiicongresso/cdrom/pdfN/386.pdf

http://www.sbmac.org.br/cnmacs/2004/cd_cnmac/files_pdf/10386a.pdf

http://www.seer.unirio.br/index.php/isys/article/view/5163/4917

http://www.bernabe.dorronsoro.es/vrp/

http://www.icmc.sc.usp.br/~sandra/G9_t2/annealing.htm

http://www.lac.inpe.br/~lorena/cap/Aula_C01.pdf

https://en.wikipedia.org/wiki/Simulated_annealing