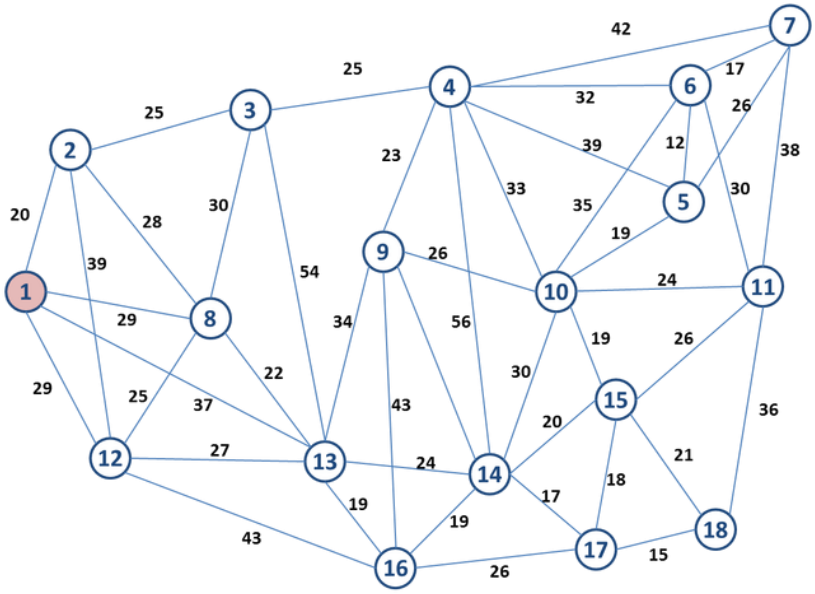
Inteligência Artificial

Thiago Henrique Leite da Silva, RA: 139920

AULA6: Exercício teórico Subida Encosta, Temp. Simulada, AG

1) (0,2) Comente sobre um problema de otimização que tem custo exponencial quando resolvido por força bruta, e nesse caso pode ser resolvido com tempera simulada/AG.

Problema: Caixeiro Viajante.



O Caixeiro Viajante é um famoso problema da computação cujo objetivo é descobrir a melhor rota para se visitar uma série de cidades e retornar ao ponto de origem, ou seja, devemos descobrir, dado um conjunto C de cidades, o ciclo hamiltoniano de menor custo. Este problema é classificado como NP completo, ou seja, não se sabe um algoritmo polinomial que encontre sua solução.

Podemos resolvê-lo por força bruta, para isso teríamos que buscar todas as possíveis soluções, comparar seus custos, e assim encontrar o caminho com o menor custo. Porém, o algoritmo ficaria rapidamente inviável de ser resolvido nos computadores que temos atualmente, já que tem tempo equivalente a ordem de **n!**. Podemos analisar na tabela deste site como cresce de uma forma extremamente rápida o número de caminhos possíveis e o tempo que levaríamos para mapeá-los:

n	rotas por segundo	(n - 1)!	cálculo total
5	250 milhoes	24	insignific
10	110 milhoes	362 880	0.003 seg
15	71 milhoes	87 bilhoes	20 min
20	53 milhoes	1.2×10^{17}	73 anos
25	42 milhoes	6.2×10^{23}	470 milhoes de anos

Em relação ao Algoritmo Genético, ele segue o modelo de seleção natural proposto por Darwin no século XIX, que em resumo, diz que as espécies mais adaptadas ao ambiente são mais propensas a sobreviver e gerar novos indivíduos. Portanto, temos duas questões a serem analisadas, definir um critério de avaliação das espécies, para selecionarmos sempre as melhores; e definir um método de cruzamento entre as melhores espécies, que chamaremos de crossover, que nada mais é do que a combinação de genes de dois indivíduos para se gerar um terceiro indivíduo. Além disso, existe um outro fato a se levar em consideração, as mutações, que são as alterações de genes aleatórios que vão variar a espécie, isso impede de chegarmos a uma região de platô em nossa população, em que não evoluímos mais, ou demoramos muito para evoluir extremamente pouco.

Inserindo esses conceitos na computação, temos que ter uma definição bem pensada do problema ao qual estamos trabalhando, para que nos aproveitemos das vantagens que este algoritmo nos proporciona.

Natureza	**Algoritmos Genéticos**
Cromossoma	Palavra binária, vetor, etc.
Gene	Caractere
Alelo	Valor do caractere
Loco	Posição na cadeia, vetor
Genótipo	Estrutura
Fenótipo	Estrutura submetida ao problema
Indivíduo	Solução
Geração	Ciclo

Demonstraremos a aplicação de uma variante do problema que estamos analisando, o caixeiro viajante, onde temos 20 cidades a serem percorridas e um ponto de partida.

Podemos definir nosso problema da seguinte maneira:

- Gene: cidade;
- Indivíduo: rota única que soluciona o problema;
- População: conjunto de indivíduos, ou seja, um conjunto de rotas;
- Pais: duas rotas que dão origem a uma nova, a partir do crossover;
- Função de fitness: função que verifica qual a melhor rota (de menor custo);
- Mutação: troca de uma cidade por outra em uma rota.

De início, definimos uma população inicial aleatoriamente, ou seja, um conjunto de N rotas que resolvem o problema, o valor N é definido pelo programador.

Após termos gerado a população inicial, vamos passar cada uma delas pela função de fitness, para medir seu desempenho, e ordená-las em ordem decrescente, do melhor desempenho para o pior, logo em seguida, eliminamos as rotas de pior desempenho.

Agora, precisamos realizar o crossover das melhores rotas para se obter filhos, esses filhos podem sofrer mutação de acordo com uma probabilidade de mutação definida pelo desenvolvedor, assim, estamos criando variações da nossa população como uma tentativa de descobrir rotas melhores.

A partir daí, é só repetirmos o processo para as próximas gerações.

- Medir desempenho da população;
- Selecionar os melhores indivíduos;
- Fazer o crossover dos mesmos;
- Obter os filhos (novas rotas);
- Realizar a mutação de acordo com a probabilidade;
- Ir para a próxima geração.

Faremos isso até encontrar a solução ótima (se esta for conhecida), ou até chegarmos a uma geração determinada pelo usuário.

Assim, conseguimos diminuir o tempo que este algoritmo levaria para ser solucionado por força bruta (73 anos) de maneira formidável. Obtendo uma solução extremamente interessante, ou até mesmo, ótima.

Fonte: O Problema do caixeiro viajante através de algoritmo genético.

2) (0,2) Descreva os principais diferenciais entre AG e os métodos tradicionais de busca (busca cega e heurística).

A maior diferença do algoritmo genético, para os métodos tradicionais, é o crossover e mutação. O algoritmo genético de fato tende a perpetuar a melhor espécie, ou seja, aperfeiçoar cada vez mais as soluções.

O fato de descartarmos as piores soluções pode até ser visto no algoritmo A*, com uma heurística aceitável, porém um ponto chave que estes algoritmos não possuem é a possibilidade de alterarmos uma solução, como uma maneira de tentarmos melhorá-la; geralmente, ao invés de mexer nas soluções encontradas, os outros algoritmos vão buscar fazendo uma varredura desde o ponto inicial, então este poder de realizar mutações, e verificar se elas foram boas com a função de fitness, dá um ganho gigantesco para estes algoritmos genéticos.

Além disso, o crossover é extremamente interessante, e um ponto de total diferencial, a combinação de duas soluções podem nos ajudar a chegar de maneira muito mais promissora a uma melhor solução do que as já mapeadas.

Se tratando do algoritmo genético, de maneira resumida e elencada, os principais pontos que podemos destacar e que são seus diferenciais são os seguintes:

- Eles buscam uma população, ou seja, um conjunto de soluções, e não uma única;
- Não precisam conhecer o problema inicialmente, o que precisamos saber é como diferenciar os resultados que iremos obter (diferente do A*);
- Usa bastante a probabilidade, diferente dos outros algoritmos que já estão mais baseados em fatos determinísticos.

Um ponto de total diferença da busca cega para a o AG, é que diferentemente da busca cega, que não sabe nada sobre seus sucessores, qual o melhor ou qual o pior, o AG está constantemente analisando os sucessores para saber com os melhores e mais aptos ao crossover.

3) (0,2) Seja o problema de encontrar o valor máximo da função $f(x) = x^2$, x inteiro. Podemos representar as soluções do problema através de um cromossomo de 6 bits.

Cromossomo	x	F(x)
0 0 0 0 1 0	2	4
0 0 1 0 0 1	9	81
0 0 0 1 0 0	4	16
0 1 0 0 0 0	16	256
0 0 0 1 1 0	6	36
0 0 0 0 0 1	1	1

Execute o passo-a-passo para três iterações de um AG, selecione 4 pares com maior função de avaliação para reprodução, aplique crossover no ponto 3 e mutação em ao menos 1 bit por estado (posição aleatória).

Passo inicial, definiremos os pares e realizaremos o crossover:

1ª Geração

- 2 e 9

000010 + 001001

=

000001

001010

- 4 e 16

000100 + 010000

=

000000

010100

- 6 e 1

000110 + 000001

=

000001

000110

Chegamos, portanto, nos seguintes indivíduos:

1. **000001 (01)**
2. **001010 (10)**
3. **000000 (00)**
4. **010100 (20)**
5. **000001 (01)**
6. **000110 (06)**

Ordenando-os, temos:

1. **010100**
2. **001010**
3. **000110**
4. **000001**
5. **000001**
6. **000000**

Agora, eles passarão pela mutação, ou seja, um dos genes será alterado de forma aleatória:

1. **010101**
2. **101010**
3. **010110**
4. **001001**
5. **100001**
6. **000100**

Feita a mutação, podemos partir para a segunda geração realizando um novo crossover:

2ª Geração

- 1 e 2

010101+ 101010

=

010010

101101

- 3 e 4

010110 + 001001

=

010001

001110

- 5 e 6

100001 + 000100

=

100100

000001

Chegamos, portanto, nos seguintes indivíduos:

1. **010010 (18)**
2. **101101 (45)**
3. **010001 (17)**
4. **001110 (14)**
5. **010001 (17)**
6. **000001 (01)**

Ordenando-os, temos:

1. **101101**
2. **010010**
3. **010001**

4. **010001**
5. **001110**
6. **000001**

Agora, eles passarão pela mutação, ou seja, um dos genes será alterado de forma aleatória:

1. **111101**
2. **010000**
3. **110001**
4. **010101**
5. **101110**
6. **001111**

Feita a mutação, podemos partir para a segunda geração realizando um novo crossover:

3ª Geração

- 1 e 2

111111+ 010000

=

111000

101101

- 3 e 4

110001 + 010101

=

110101

010001

- 5 e 6

101110 + 001111

=

101111

110001

Chegamos, portanto, nos seguintes indivíduos:

1. **111000 (56)**
2. **101101 (45)**
3. **110101 (53)**
4. **010001 (17)**
5. **101111 (47)**
6. **110001 (49)**

Ordenando-os, temos:

1. **111000 (56) => f(x) = 3136**
2. **110101 (53) => f(x) = 2809**
3. **110001 (49) => f(x) = 2401**
4. **101111 (47) => f(x) = 2209**
5. **101101 (45) => f(x) = 2025**
6. **010001 (17) => f(x) = 289**

Por fim, podemos notar que após três gerações em nosso algoritmo genético, nossa população melhorou muito comparado com a geração inicial, em sua função de maximixar a função $f(x) = x^2$.

4) (0,4) Pesquise um trabalho/artigo usando têmpera simulada para resolver um problema real. Descreva:

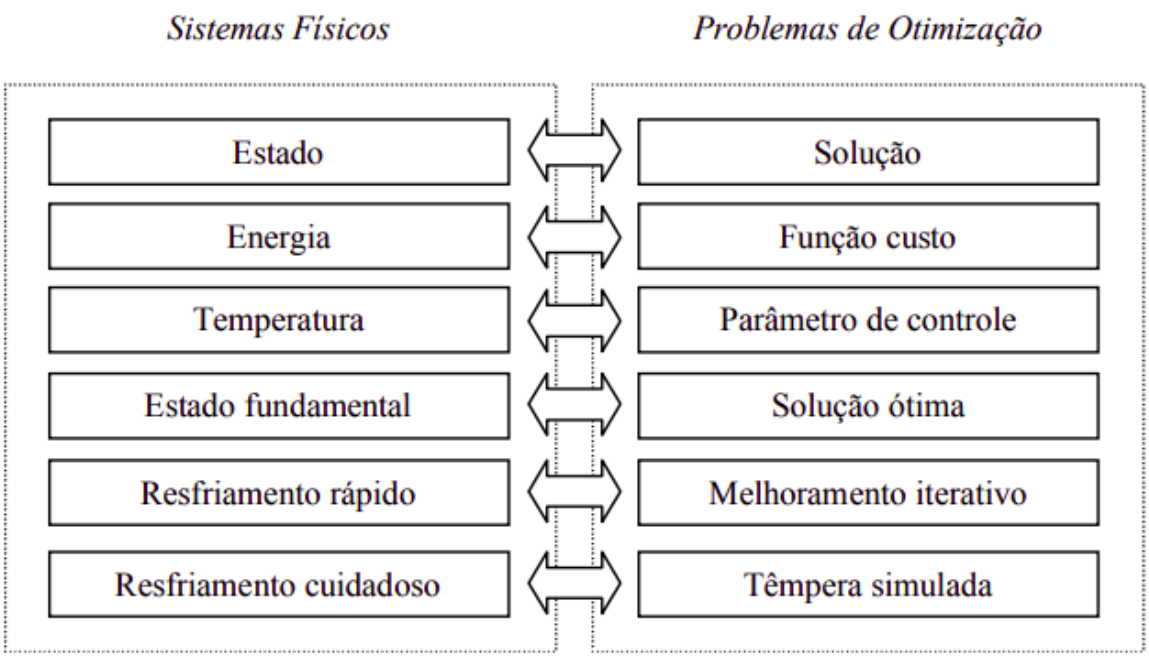
Qual o problema abordado.

O problema que encontrei que usa têmpera simulada para ser resolvido, é o escalonamento com restrição de recursos com múltiplos modos de processamento, que se trata de uma forma de otimização do tempo e alocação de recursos para finalizar todas as tarefas estipuladas, identificando o modo e tempo de início de processamento de cada uma das tarefas.

Com base nas tarefas determinadas e nas devidas restrições relacionada aos recursos, temos que minimizar o comprimento do escalonamento, que seria o tempo de processamento das tasks. Apesar de a priori não parecer, este problema é bem difícil de ser resolvido e entra na classe dos problemas NP-Difíceis; portanto, como forma de melhorar a obtenção de bons resultados na resolução do mesmo, veremos sua abordagem com o algoritmo de têmpera simulada.

Baseado em: http://www.din.uem.br/sbpo/sbpo2002/pdf/arq0111.pdf

Quais os parâmetros do modelo:



a) número de elementos,

Para realização dos testes do algoritmo na resolução do problema, foram definidas 5 instâncias com 10 tarefas, 3 instâncias com 16 tarefas e 7 instâncias com 20 tarefas, totalizando 15 instâncias. Em cada uma delas, existe a alocação de dois recursos renováveis e dois não renováveis, ou seja, estamos utilizando a limitação de recursos; além disso, cada uma dessas tarefas pode ser realizada de três maneiras diferentes.

Em resumo, iremos testar 15 casos, onde o algoritmo precisa nos retornar o menor tempo de processamento da quantidade de tarefas especificada, levando em conta recursos não renováveis, limitados, e também os diferentes modos que possuímos de realizar a tarefa.

b) função de energia a ser otimizada

No artigo, não cita especificamente a função de energia, apenas foi definido que um novo estado S' só é aceito se seu custo for melhor que o estado S, sendo que:

$$S (= C(S´) – C(S) 0)$$

Onde S é a solução, S' a possível nova solução e C a função de custo, que leva em conta os seguintes elementos: P, T, M e R.

P = tempo de processamento;

T = quantidade de tarefas;

M = modo de processamento;

R = conjunto de recursos;

c) número de execuções até minimizar a temperatura/função,

Foram em média 15 execuções para se chegar aos resultados abaixo.

As colunas TFPSPLIB e TFTS são referentes, respectivamente, a solução ótima e os tempos de finalização apresentados pelo algoritmo têmpera simulada:

Instância	$\|T\|$	TF_{PSPLIB}	TF_{TS}	d_a	d_r	t(s)
J102_2	10	20	20	0	0	0.32
J104_3	10	23	23	0	0	0.17
J1028_8	10	16	26	0	0	0.12
J1048_5	10	14	14	0	0	0.17
J1051_5	10	29	29	0	0	0.16
J1620_3	16	28	28	0	0	0.35
J1642_2	16	29	30	1	3.4	3.3
J1653_8	16	25	25	0	0	1.37
J2011_10	20	26	26	0	0	1.17
J2025_5	20	30	31	1	3.3	1.46
J2045_6	20	36	38	2	5.5	6.6
J2051_5	20	28	28	0	0	2.42
J2051_6	20	22	23	1	4.5	1.44
J2062_10	20	26	28	2	7.7	1.38
J2063_7	20	36	36	0	0	3.26

d) solução final

De acordo com a tabela acima, o algoritmo de têmpera simulada atingiu das 15 instâncias de teste, 10 soluções ótimas, e a média de desvio foi de 4.9% nas outras 5. Ou seja, o algoritmo teve um desempenho muito bom, e segundo os autores do artigo, o algoritmo pode ser aperfeiçoado através da exploração do domínio do problema.

No geral, o algoritmo performou extremamente bem e chegou nas soluções de uma maneira rápida e eficiente.