Universidade Federal do Rio Grande do Sul - UFRGS
Instituto de Informática

INF05010-A - Optimização Combinatória - 2014/2

Problema de viagens para torneio com algorítimo genético

(Traveling Tournament Problem with Genetic Algorithm)

Prof. Marcus Rolf Peter Ritt

Discentes autores:
Pedro Henrique Frozi - 161502
Lucas Herbert Jones - 124631

24 de novembro de 2014

1. Introdução

Apresentamos nesse relatório o Traveling Tournament Problem (TTP), um relato do desenvolvimento da formulação, um algorítimo genético e os resultados na busca de soluções ótimas tanto através de um solver quando do algorítimo genético.

Neste trabalho apresentaremos uma formulação para o problema *Travelling Tournament Problem (TTP)* em programação inteira, mostrando como foram definidas as variáveis da formulação, como obtivemos a função objetivo e também a definição das restrições do problema. Através desta formulação realizada, desenvolvemos um algoritmo genético que pudesse encontrar uma solução para o problema, utilizando os conceitos desta metaheurística vistos na disciplina.

2. O Problema

O *Traveling Tournament Problem* (TTP) consiste na busca de uma solução que minimize a distância total das viagens necessárias entre as equipes de uma determinada liga ou torneio, tendo em vista que estes times devem jogar em uma vez em casa e outra fora de casa conta cada time.

Algumas outras restrições deverão ser observadas:

- Número de times é obrigatoriamente um número par;
- Dois times não podem se enfrentar em duas rodadas consecutivas, ou seja, caso em uma rodada o time A enfrente o time B, na próxima rodada estes times não poderão se enfrentar novamente;
- Nenhum time poderá jogar mais de três jogos consecutivos em casa;
- Nenhum time poderá jogar mais de três jogos consecutivos fora de casa;
- Em jogos fora consecutivos, as viagens são feitas diretamente para o próximo local sem a necessidade de voltar para casa;
- Todos os times iniciam em casa e devem terminar a campeonato em casa;

3. Formulação do Problema

Iniciamos a formulação identificando as variáveis necessárias para a construção de uma solução. Precisaríamos de uma variável binária de seleção “X”, que quando fosse igual a 1, identificaria a seleção de uma viagem entre um time e uma localidade.

Além disso precisávamos de uma forma de referenciar aos times de forma numérica. Definimos então duas variáveis “i” e “j”, representando times e variando de 1 até “n”, sendo “n” o número total de times. Como cada time tem sua casa, utilizamos a mesma numeração usada nos times para as localidades, sendo “i” um time e uma localidade (a casa desse time). Sendo assim, definimos que X(i, j) seria a seleção de um jogo entre o time “i” e o time “j” na localidade “i”. Ou seja, o primeiro parâmetro da função X, representa o time que joga em casa.

Faltava ainda a noção de tempo a essa formulação, pois é necessário identificar que o time “i” joga em casa contra o time “j” em um dado momento no tempo. Para isso criamos uma variável “k” que representa uma rodada e varia de 1 até “Kn”, sendo “Kn” igual a 2(n-1), valor correspondente ao número de rodadas do campeonato. Com isso temos a uma variável booleana de seleção X(i, j, k), sendo X a variável que indica se o time “i” joga em casa contra o time “j” na rodada “k”.

Uma formulação de função objetivo, pelo que definimos até aqui, seria a soma de todos valores de A(i, j) multiplicado por X(i, j, k) de todos os possíveis valores para i, j e k. Onde “A” é uma matriz bidimensional que contem as distâncias entre duas localidades. Ou seja, os valores de distância da matriz A só são considerados quando o valor de X(i, j, k) for igual a 1.

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{2(n-1)} a_{ij} * x_{ijk}$$

Nessa formulação inicial somente estaríamos considerando a ida do time j de sua casa até a localidade i, porém desconsiderando a volta para casa e/ou possíveis viagens diretas de fora de casa para outro local fora de casa.

Essa formulação não só esta incompleta, como também lhe faltam elementos para representar o problema. Na definição do problema existe a possibilidade de viagens de fora de casa para outra localidade fora de casa. Para que um time que joga fora de casa na rodada k, viaje para outra cidade fora de casa na rodada k+1, é necessário representarmos a localidade na rodada anterior. Para isso precisaríamos de mais uma variável “l”, novamente para representar um time e uma localidade, e de um novo elemento X na função objetivo.

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{l=1}^{n} \sum_{k=1}^{2(n-1)} a_{li} * x_{ijk} * x_{li(k-1)}$$

Esse novo elemento X, ao final da função objetivo, indica que o time “l” jogou com o time “i” na rodada anterior. Logo só serão somados os valores de distancia entre “l” e “i” quando “i” jogar com “j” na rodada k e quando “l” e “i” jogarem na rodada anterior. Além disso com a introdução da variável com valor k-1 e dado que não existe rodada k=0, é preciso adicionar um novo somatório a F.O. para tratar a primeira rodada exclusivamente e iniciar a atual formulação em k=2.

Ainda falta lidar com a questão dos times terem que retornarem para casa após a ultima rodada. Para isso seria adicionado mais um somatório independente na função objetivo.

Apesar de essa formulação esta chegando próxima a uma formulação de função objetivo completa para o problema, ela esta multiplicando duas variáveis X, o que se configura como uma função quadrática e não-linear. Para que possamos

resolver esse problema de forma linear, precisamos “linearizar” a F.O.

Uma técnica para linearizar a função objetivo é adicionar um nova variável no lugar de X e adicionando algumas restrições. No caso adicionamos três novas variáveis: “y”, “z” e “w”.

Mostramos a seguir a formulação completa do problema, já linearizada:

$$min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}\sum_{k=2}^{Kn} a_{ij} * y_{ijlk} + a_{il} * z_{ijlk} + a_{li} * w_{ijlk} + \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} * x_{ij1} + \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} * x_{ijKn}$$

Sujeito as seguintes restrições:

Restrições para a “linearização” do problema:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}\sum_{k=2}^{Kn} y_{i,j,l,k} >= x_{i,j,k} + x_{l,i,k-1} - 1$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}\sum_{k=2}^{Kn} z_{i,j,l,k} >= x_{i,j,k} + x_{l,j,k-1} - 1$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}\sum_{k=2}^{Kn} w_{i,j,l,k} >= x_{i,j,k} + x_{j,l,k-1} - 1$$

Time não joga contra si mesmo:

$$\sum_{k=1}^{Kn} x_{ijk} = 0 \quad \forall i \in [n], \forall j \in [n], i = j$$

Número fixo de jogos por rodada:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = n/2 \quad \forall k \in [Kn]$$

Todo o jogo de i x j (par de times) ocorre apenas 1 vez:

$$\sum_{k=1}^{Kn} x_{ijk} = 1 \quad \forall i \in [n], \forall j \in [n], i <> j$$

Em uma rodada em que i joga com j, tanto i quanto j NAO podem jogar outros jogos na mesma rodada:

$$\sum_{j=1}^{n} x_{ijk} + x_{jik} = 1 \quad \forall i \in [n], \forall k \in [Kn]$$

Todo time joga número Kn/2 de vezes em casa:

$$\sum_{j=1}^{n}\sum_{k=1}^{Kn} x_{ijk} = Kn/2 \quad \forall i \in [n]$$

Todo time joga número Kn/2 de vezes como visitante:

$$\sum_{i=1}^{n}\sum_{k=1}^{Kn} x_{ijk} = Kn/2 \quad \forall j \in [n]$$

Sequências de mais de três jogos fora de casa não devem acontecer:

$$\sum_{j=1}^{n} x_{ijk} + x_{ij(k+1)} + x_{ij(k+2)} + x_{ij(k+3)} \leq 3 \quad \forall i \in [n], \forall k \in [Kn-3]$$

Sequências de mais de três jogos em casa não devem acontece:

$$\sum_{i=1}^{n} x_{ijk} + x_{ij(k+1)} + x_{ij(k+2)} + x_{ij(k+3)} \leq 3 \quad \forall j \in [n], \forall k \in [Kn-3]$$

Dois times não podem se enfrentar em rodadas consecutivas:

$$x_{ijk} + x_{ji(k+1)} \leq 1 \quad \forall i \in [n], \forall j \in [n], \forall k \in [Kn-1]$$

3.1| Representação de uma Solução

A representação de uma solução, para a formulação proposta, é composta por um grupo de de itens x(i,j,k), y(i,j,l,k), z(i,j,l,k) e w(i,j,l,k) que para determinados valores de i, j, l e k recebem o valor 1. Por exemplo em uma liga composta por quatro times, sendo esses ATL, NYM, PHI, MON e dada uma matriz A de distâncias entre eles, podemos obter uma solução com o seguinte conjunto de variáveis com valores iguais a 1 (valores organizados por rodadas para melhor visualização):

1	x[NYM,PHI,1]	x[MON,ATL,1]
2	x[NYM,ATL,2]	x[MON,PHI,2]
3	x[NYM,MON,3]	x[PHI,ATL,3]
4	x[ATL,NYM,4]	x[PHI,MON,4]
5	x[ATL,MON,5]	x[PHI,NYM,5]
6	x[ATL,PHI,6]	x[MON,NYM,6]

1				
2		z[NYM,ATL,MON,2]	z[MON,PHI,NYM,2]	
3	y[PHI,ATL,MON,3]	z[PHI,ATL,NYM,3]		w[NYM,MON,PHI,3]
4	y[ATL,NYM,PHI,4]	z[PHI,MON,NYM,4]		w[ATL,NYM,MON,4]
5		z[ATL,MON,PHI,5]	z[PHI,NYM,ATL,5]	
6	y[MON,NYM,ATL,6]	z[MON,NYM,PHI,6]		w[ATL,PHI,NYM,6]

Esses mesmos dados podem ser apresentados em uma tabela onde as linhas são rodadas e as colunas são os times. Os valores internos da tabela, representam quem joga contra o time da coluna na rodada da linha. O simbolo “@” antes da sigla do time indica que esse time joga em casa, nessa rodada. Segue um exemplo desse formato de tabela:

	ATL	**NYM**	**PHI**	**MON**
1	@MON	PHI	@NYM	ATL
2	@NYM	ATL	@MON	PHI
3	@PHI	MON	ATL	@NYM
4	NYM	@ATL	MON	@PHI
5	MON	@PHI	NYM	@ALT

6	PHI	@MON	@ATL	NYM

4| Algorítimo Genético

A metaheuristica ultilizada será *Genetic Algorithms* (GA), baseado na evolução natural das espécies. Para aplicação desta técnica, primeiramente precisamos definir alguns fundamentos relacionados com a biologia:

1. O Gene: Determina qual o tipo de representação do elemento de determinada característica da solução. Neste caso, o tipo que nós utilizaremos será {0,1};
2. Alelo: Representa a instância de um gene. O alelo será representado pelo valor da variável x_{ijk};
3. Cromossomo: Conjunto de genes que determinam todas as características de determinado indivíduo. O nosso cromossomo é um array de bits, onde cada um dos bits representa um alelo. O tamanho do cromossomo é definido por: *n*n*2(n-1).*

   Que corresponde a todas as variáveis x_{ijk} do nosso problema;

```
Estrutura do Cromossomo (4 times)
012...                                                                                                 ...95
XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX
|k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5| |k0a5|
| j=0| | j=1| | j=2| | j=3| | j=0| | j=1| | j=2| | j=3| | j=0| | j=1| | j=2| | j=3| | j=0| | j=1| | j=2| | j=3|
|           i=0           | |           i=1           | |           i=2           | |           i=3           |
```

4. População: Conjunto de indivíduos que carregam as suas características em seu cromossomo. Para o nosso trabalho, cada um dos indivíduos representa uma solução para o problema TTP definido;
5. Aptidão: Valor atribuído ao indivíduo conforme a qualidade das suas características genéticas. Portanto, aptidão equivale a função objetivo do nosso problema.

A “seleção natural” deverá ser realizada com base na população de indivíduos que foi avaliada. A ideia consiste em verificar quais foram os melhores indivíduos da população conforme o valor de aptidão obtido, então, a partir destes indivíduos “mais qualificados”, gerar a próxima população. Os pais selecionados gerarão três

Existem algumas técnicas para lidar com as restrições de um problema quando utilizamos algoritmos genéticos, entre elas reparação, penalização e representação específica. No trabalho aqui proposto utilizaremos a técnica de penalização, que consiste em adicionar as chamadas funções de penalização Φq à nossa função de aptidão. Quanto mais distante a solução estiver do espaço de soluções factíveis, menor a qualidade da nossa função de aptidão.

<u>Parâmetros de Entrada</u>

Os parâmetros de entrada utilizados pelo nosso algoritmo serão:

***AlgoritmoGenetico(t, M, T, c, p, d, m, s)**, onde:*

t - Quantidade de times,

M - Matriz que representa uma instância do problema,

T - Tamanho da população de entrada,

c - Taxa de filhos gerados a partir do crossover na próxima geração,

p - Taxa de filhos gerados a partir da operação de mutação na próxima geração,

m - Taxa de mutação que ocorrerá dentro do cromossomo,

d - Quantidade máxima aceitável de gerações sem melhora (para o critério de parada),

s - Tempo máximo de execução (para o critério de parada).

4.1| Solução Inicial

Neste algoritmo, a nossa solução inicial representa uma população de soluções composta de *T* indivíduos, onde cada um leva as suas "características". Esses indivíduos serão gerados aleatoriamente, mas respeitando algumas condições minimas para possibilitar soluções válidas. Por exemplo, ao gerar as soluções, não são aceitas alelos que indiquem que um time joga contra ele mesmo. Da mesma forma buscamos eliminar já na solução inicial a possibilidade de um time jogar dois jogos na mesma rodada. Assim temos soluções com maior chance de validade, bem como bons candidatos a seleção e por consequência um algorítimo mais eficiente.

4.2| Mutação, Combinação e Elitismo

A nova geração de indivíduos será criada a partir de três tipos de operações: Mutação, Combinação e Elitismo. A operação de **Mutação** no nosso algoritmo consiste em selecionar aleatoriamente uma determinada rodada k e embaralha-lá, fazendo com que diferentes times se enfrentem na rodada k do individuo mutado. Portanto, uma mutação simples realiza um alteração apenas nos jogos de determinada rodada. Já a **Combinação** cria um novo indivíduo que, dados dois pais selecionados aleatoriamente, possui exatamente a metade dos alelos de um individuo pai, e a outra metade de outro. Finalmente, o **Elitismo** faz com que os melhores indivíduos da geração atual tenham os seus cromossomos passados para a próxima geração como cópias para outros indivíduos.

Para gerar a próxima geração de indivíduos, quatro parâmetros passados na chamada do programa são de extrema importância:

-***T n***: Indica que a população de entrada do algoritmo terá o tamanho ***n***. Todas as populações das próximas gerações também terão este mesmo tamanho;

-***p r***: Indica a taxa ***r*** de indivíduos das próximas gerações que serão resultantes de uma operação de mutação;

-***m r***: Indica a probabilidade de determinada rodada (grupo de alelos de um indivíduo) de sofrer a operação de mutação.

Quanto maior o valor de ***r***, maior a probabilidade de que uma determinada rodada do individuo selecionado sofra a mutação;
-***c r***: Indica a taxa ***r*** de indivíduos das próximas gerações que serão resultantes de uma operação de mutação.

4.3| Critério de Parada

Serão utilizados os seguintes critérios de parada para o algoritmo:

- Quantidade de Gerações sem Melhora: O critério de parada utilizado para o problema consiste em verificar a quantidade de Gerações sem melhora na aptidão. Esta quantidade será um dos parâmetros de entrada do nosso algoritmo;

- Tempo: Parâmetro de entrada que determina o tempo máximo de execução do algoritmo;

4.4| Fitness

A função de fitness é a responsável por calcular quão adaptado esta o individuo aos critérios estabelecidos. No nosso caso, criamos um grupo de penalidades a essa função fitness quando o individuo não tiver uma solução válida. Após passar por todas as validações, o valor da função fitness é igual ao da função objetivo.

As funções abaixo são funções de validação que retornam 0 caso a solução seja valida, ou 1 caso não seja. Caso a solução seja invalida, o valor do fitness é ampliado pelo valor da potencia de dez mostrado abaixo. Os diferentes pesos relacionados a cada validação, cria uma lista de prioridades para o calculo do fitness de um individuo. É importante fazer uma boa seleção de pesos para ajudar na convergência das soluções no sentido desejado.

```
fitness += playYourself()                * pow(10,17);
fitness += matchsPerRound()              * pow(10,16);
fitness += validateUniqueGame()          * pow(10,15);
fitness += maxThreeGamesHome()           * pow(10,14);
fitness += maxThreeGamesOut()            * pow(10,13);
fitness += playEachOtherAgain()          * pow(10,12);
fitness += oneGamePerTeamPerRound()      * pow(10,11);
fitness += funcaoObjetivo();
```

5.Resultados

Instância	-T	-p	-m	-c	-d	Tempo(seg)	Total Ger.	Fitness	Solução Ótima	% Otim.
NL4	12	0.3	0.1	0.3	3	0.002785	4	8559	8276	96.69%
super4	12	0.3	0.1	0.3	3	0.001533	4	89317	63405	70.99%
super4	12	0.3	0.1	0.3	15	0.005872	16	89317	63405	70.99%
super4	12	0.3	0.1	0.3	50	0.022853	51	89317	63405	70.99%
super4	12	0.3	0.1	0.3	1000	0.285921	1001	89317	63405	70.99%
super4	12	0.3	0.1	0.3	2000	0.561663	2001	89317	63405	70.99%
super4	30	0.3	0.1	0.3	1000	0.426776	1001	64270	63405	98.65%
super4	35	0.3	0.1	0.3	2000	1.14921	2020	63405	63405	100.00%
galaxy4	30	0.3	0.1	0.3	1000	0.801635	1786	437	416	95.19%
galaxy4	30	0.3	0.1	0.3	2000	1.2361	2786	437	416	95.19%
galaxy4	60	0.3	0.1	0.3	1000	0.839437	1001	421	416	98.81%
galaxy4	60	0.3	0.1	0.3	2000	1.69821	2001	421	416	98.81%

6. Conclusões

No desenvolvimento do trabalho ficou claro que o algorítimo genético, sem restrições na geração randômica inicial e na mutação, seria inviável mesmo para conseguir soluções válidas. O TTP é um problema que precisa de soluções muito precisas para serem válidas e mesmo que se tenha uma solução válida, facilmente ela pode deixar de ser, dada uma mutação ou um *crossover*.

Constatado isso, era evidente que teríamos que buscar mutações e *crossovers* mais regrados para que possamos ter um algorítimo genético que tivesse passos incrementais.

Fizemos um bom progresso no sentido de melhorar as soluções iniciais, desenvolver as funções de validação, desenvolver mutações e *crossovers* mais regrados. Além disso conseguimos resultados ótimos para alguns problemas, o que validou nosso algorítimo pelo menos para alguns casos.

Acreditamos que nosso esforço foi recompensado, pois aprendemos muito sobre o problema TTP e sobre algorítimos genéticos.

7. Bibliografia