Título - Relatório ALGAV

Subtítulo – Sprint C

Turma DE _ Grupo 26
1181628 _ André Novo
1180782 _ Diogo Ribeiro
1170500 _ Hugo Frias
1161255 _ João Cruz
1180730 _ Vera Pinto

Professor
Jorge Coelho, JMN

Unidade Curricular
Algoritmia Avançada (ALGAV)

Data: Janeiro/2021

1. Introdução

Neste sprint C, adaptamos tudo exceto duas condições de término (tempo limite e individuo com valor da função de avaliação menor ou igual a um valor indicado), nenhuma soft constraint.

2. Representação do conhecimento do domínio de Transportes Urbanos/Intermunicipais a usar no Algoritmo Genético (AG)

- horario(Path,Trip,List_of_Time).
- workblock(WorkBlock, List_of_Trips, StartTime, EndTime).
- vehicleduty(VehicleDuty, List_of_WorkBlocks).
- lista_motoristas_nworkblocks(vehicleDutyId, [name,nWorkBlocks]).

- Cromossoma:

[5188,5188,5188,276,276,16690,16690,18107,18107,18107,18107,18107,18107]

Lista de motoristas para um determinado vehicle duty, cujo número de motoristas corresponde ao número de Workblocks que esse mesmo motorista precisa de realizar

- Agendas temporais:

Fizemos uso de agendas temporais, de forma a facilitar a resolução do AG, criando tripletos com limite superior de bloco, limite inferior de bloco e motorista, representando um intervalo de blocos consecutivos.

```
% Condensador
% Nota: Ordenando pelo Elemento, o algoritmo seria mais eficiente,
% porque não teria que percorrer toda a lista.

:-dynamic listaAux/1.

elimina([],_,[]):-!.
elimina([X|R1],L,[X|R2]):-
        not(member(X,L)),!,
        elimina(R1,L,R2).
elimina([_|R1],L,R2):-
        elimina(R1,L,R2).

condensadorDigitalE(Lista):-
   (retractall(horarioMotorista(_));true),
   (retractall(listaAux(_));true),
   asserta(listaAux([])),
   asserta(horarioMotorista([])),
   condensadorDigital1(Lista).

condensadorDigital1([]):-!.
condensadorDigital1([(LimiteInferior,LimiteSuperior,Elemento)|Lista]):-
   condensadorDigital2((LimiteInferior,LimiteSuperior,Elemento),Lista),
   retract(listaAux(X)),
   elimina(Lista,X,ListaX),
   asserta(listaAux([])),
   condensadorDigital1(ListaX).

condensadorDigital2(L,[]):-
   retract(horarioMotorista(LL)),
   asserta(horarioMotorista(([L|LL]))).
condensadorDigital2((LimiteInferior,LimiteSuperior,Elemento),[(LimiteInferior1,LimiteSuperior1,Elemento1)|Lista]):-
   (Elemento == Elemento1, LimiteSuperior ==  LimiteInferior1),
   retract(listaAux(ListaAnterior)),
   asserta(listaAux(([(LimiteInferior1,LimiteSuperior1,Elemento1)|ListaAnterior]))),
   condensadorDigital2((LimiteInferior,LimiteSuperior1,Elemento),Lista).
condensadorDigital2((LimiteInferior,LimiteSuperior,Elemento),[_|Lista]):-
   condensadorDigital2((LimiteInferior,LimiteSuperior,Elemento),Lista).
```

3. Operadores de cruzamento e mutação implementados

```
?- cruzar([5188,16690,18107,18107,276,16690,18107,16690,5188,16690,18107,16690,276],
[18107,18107,16690,276,16690,16690,16690,18107,5188,16690,276,18107,5188],2,7,L),
write('Lista: '), write(L),nl.
Lista: [16690,18107,18107,276,16690,18107,18107,5188,276,18107,5188,5188,18107]
```

Relativamente ao cruzamento, nós alterámos o método de cruzar entre dois indivíduos, aplicando restrições ao número máximo de Workblocks por motorista (gene).

4. Aleatoriedade no cruzamento entre indivíduos da população

```
gera_geracao(N,G,Pop):-
        write('Geração '), write(N), write(':'), nl, write(Pop), nl,
        random_permutation(Pop,PopulacaoPermutada),
        cruzamento(PopulacaoPermutada,NPop1),
        % mutacao(NPop1,NPop),
        % avalia_populacao(NPop1,NPopAv),
        guardaDigital(NPop1),
        penalizacoes(NPopAv),
        ordena_populacao(NPopAv,NPopOrd),
        N1 is N+1,
        append(Pop,NPopOrd,ListaDaGeracaoAnteriorComANova),
        length(ListaDaGeracaoAnteriorComANova,TamanhoETal),
        QuantidadeDeMelhoresIndividuos is TamanhoETal/2,
        junta_geracoes(ListaDaGeracaoAnteriorComANova,ListaOrdenadaCrescentemente,QuantidadeDeMelhoresIndividuos),
        append(ListaOrdenadaCrescentemente,ListaDaGeracaoAnteriorComANova,AuxProximaGeracao),
        sort(AuxProximaGeracao,AuxProximaGeracaoSemRepetidos),
        ordena_populacao(AuxProximaGeracaoSemRepetidos,ProximaGeracaoSemRepetidos),
        ProximaGeracaoSemRepetidos \== Pop,
        gera_geracao(N1,G,ProximaGeracaoSemRepetidos).
```

Na linha de código selecionada, nós criamos uma permutação pseudorrandómica de modo a existir aleatoriedade entre os indivíduos.

5. Restrições consideradas (hard e soft constraints) e implementação do cálculo da função de avaliação

- Restrição de descanso de 1 hora depois de um bloco de 4 horas: verificar se entre blocos de 4 horas e o bloco seguinte na agenda do motorista existe 1 hora ou mais
- Restrição de não ultrapassar 4 horas consecutivas: ver se nos vários tripletos de cada motorista o limite superior menos o limite inferior ultrapassa os 14400 segundos
- Quando um indivíduo não atende a uma hard constraint, os indivíduos permanecem, mas são penalizados em função das hard constraints que violam. No caso destas constraints, o peso é 10 (em segundos)

```
:-dynamic listaPenalizacao/1.

penalizacoes(ListaX):-
        horarioMotorista(M),
        (retractall(listaAux(_));true),
        asserta(listaAux([])),
        (retractall(listaPenalizacao(_));true),
        asserta(listaPenalizacao([])),
        penaliza4Horas(M,ListaResultado),
        penalizaIntervalos(M,ListaResultado1),
        append(ListaResultado,ListaResultado1,ListaResultado2),sort(ListaResultado2,ListaResultado3),
        concatPenalizacoes(ListaResultado3,ListaX).

concatPenalizacoes([],[]):-!.
concatPenalizacoes([(Custo,Motorista)|Lista],[Y|ListaResultado]):-
       member((_,Motorista),Lista),
       concatPenalizacoes2((Custo,Motorista),Lista),
       retract(listaAux(Y)),
       retract(listaPenalizacao(L)),
       asserta(listaAux([])),
       asserta(listaPenalizacao([])),
       elimina(Lista,L,ListaL),
       concatPenalizacoes(ListaL,ListaResultado).
concatPenalizacoes([(Custo,Motorista)|Lista],[(Custo,Motorista)|ListaResultado]):-
       concatPenalizacoes(Lista,ListaResultado).

concatPenalizacoes2(_,[]):-!.
concatPenalizacoes2((Custo,Motorista),[(Custo1,Motorista)|Lista]):-
      CustoX is Custo+Custo1,
      retract(listaAux(_)),
      retract(listaPenalizacao(ListaPenalizacoes)),
      asserta(listaPenalizacao(([(Custo1,Motorista)|ListaPenalizacoes]))),
      asserta(listaAux((CustoX,Motorista))),
      concatPenalizacoes2((CustoX,Motorista),Lista).
concatPenalizacoes2((Custo,Motorista),[_|Lista]):-
      concatPenalizacoes2((Custo,Motorista),Lista).

% Penalizamos fortemente um motorista que trabalhe mais que 14400
% segundo de forma consecutiva.

penaliza4Horas([],[]):-!.
penaliza4Horas([(LimiteInferior,LimiteSuperior,Motorista)|Lista],[(Valor,Motorista)|ListaResultado]):-
    TempoTrabalho is LimiteSuperior - LimiteInferior,
    (TempoTrabalho > 14440),Valor is (TempoTrabalho - 14440)*10,
    penaliza4Horas(Lista,ListaResultado).
penaliza4Horas([(_,_,Motorista)|Lista],[(0,Motorista)|ListaResultado]):-
        penaliza4Horas(Lista,ListaResultado).
```

```
% Penalizamos fortemente um motorista que não realize uma pausa de, pelo
% menos, 3600 segundos após um bloco de 4h de trabalho.

penalizaIntervalos([X|Lista],ListaResultado):-
       penalizaIntervalos1(X,Lista,ListaResultado).
penalizaIntervalos1(_,[],[]):-!.
penalizaIntervalos1((LimiteInferior,LimiteSuperior,Motorista),[(LimiteInferior1,LimiteSuperior1,Motorista1)|Lista],[(Valor,Motorista)|ListaResultado]):-
    (Motorista == Motorista1),
    Tempo is LimiteSuperior - LimiteInferior,
    Intervalo is LimiteInferior1 - LimiteSuperior,
    (Tempo>= 14440, Intervalo<3600),
    Valor is (Intervalo*8),
    penalizaIntervalos1((LimiteInferior1,LimiteSuperior1,Motorista1),Lista,ListaResultado).
penalizaIntervalos1((_,_,Motorista),[(LimiteInferior1,LimiteSuperior1,Motorista1)|Lista],[(0,Motorista)|ListaResultado]):-
    penalizaIntervalos1((LimiteInferior1,LimiteSuperior1,Motorista1),Lista,ListaResultado).
```

Relativamente ao algoritmo, geramos listas com pares constituídos por um motorista e o respetivo peso. No final, usámos um predicado criado por nós, que junta os pares de um mesmo motorista num único par, com o peso a igualar a soma do peso de todos os pares desse mesmo motorista.

6. Seleção da nova geração da população

O grupo não fez uso do predicado mutação e durante o processo de seleção dos indivíduos que iriam passar para a próxima geração, selecionámos metade da quantidade permitida de indivíduos considerados "elite" e a restante metade de sujeitos menos bons de modo a existir uma maior variedade aquando do cruzamento.

```
gera_geracao(G,G,Pop):-!,
        write('Geração '), write(G), write(':'), nl, write(Pop), nl,
        melhor_individuo(Pop).
gera_geracao(N,G,Pop):-
        write('Geração '), write(N), write(':'), nl, write(Pop), nl,
        random_permutation(Pop,PopulacaoPermutada),
        cruzamento(PopulacaoPermutada,NPop1),
        % mutacao(NPop1,NPop),
        % avalia_populacao(NPop1,NPopAv),
        guardaDigital(NPop1),
        penalizacoes(NPopAv),
        ordena_populacao(NPopAv,NPopOrd),
        N1 is N+1,
        append(Pop,NPopOrd,ListaDaGeracaoAnteriorComANova),
        length(ListaDaGeracaoAnteriorComANova,TamanhoETal),
        QuantidadeDeMelhoresIndividuos is TamanhoETal/2,
        junta_geracoes(ListaDaGeracaoAnteriorComANova,ListaOrdenadaCrescentemente,QuantidadeDeMelhoresIndividuos),
        append(ListaOrdenadaCrescentemente,ListaDaGeracaoAnteriorComANova,AuxProximaGeracao),
        sort(AuxProximaGeracao,AuxProximaGeracaoSemRepetidos),
        ordena_populacao(AuxProximaGeracaoSemRepetidos,ProximaGeracaoSemRepetidos),
        ProximaGeracaoSemRepetidos \== Pop,
        gera_geracao(N1,G,ProximaGeracaoSemRepetidos).

melhor_individuo([X|_]):- nl, write('Melhor Solução: '), write(X), nl, !.

junta_geracoes(AmbasAsPopulacoes,SubListaSemHs,NumeroElementos):-
       ordena_populacao(AmbasAsPopulacoes,AmbasAsPopulacoesOrdemCrescente),
       sublista(AmbasAsPopulacoesOrdemCrescente,1,NumeroElementos,SubListaFormada),
       delete(SubListaFormada,h,SubListaSemHs).
```

7. Parametrização da condição de término do AG

Nomeadamente às condições de término do AG, o nosso grupo conseguiu implementar as seguintes:

- Nº de gerações limite
- Estabilização total da população

8. Análise da evolução do valor da função de avaliação do melhor indivíduo e da média de todos os indivíduos de cada geração do AG

Relativamente a este tópico, o nosso grupo teve problemas aquando da conjugação de todos os blocos de código desenvolvidos nos pontos anteriores, pelo que não conseguimos obter resultados para analisar. Conseguimos obter soluções de gerações individualmente, mas não proceder de forma automática à obtenção automática de resultados após X gerações.

9. Proposta de duas heurísticas rápidas de criação de uma solução

Relativamente às heurísticas, propomos a seguinte: atribuir ao motorista Workblocks consecutivos que não ultrapassem 4 horas, por forma a garantir que, num determinado bloco, o motorista não excede o limite máximo de 4 horas de trabalho consecutivo, que equivale a uma das hard constraints implementadas.

10. Conclusões

Após este Sprint C, o grupo não conseguiu concluir resultados gerais relativamente à adaptação feito no algoritmo genérico para o sequenciamento de motoristas num autocarro. No entanto, conseguimos testar certas partes do algoritmo genético, de modo a assimilar os conceitos pretendidos com a realização deste Sprint.

Quanto à divisão de tarefas entre elementos do grupo decidimos que iriamos todos participar na adaptação do algoritmo. Deste modo, um de nós estava a fazer partilha de ecrã enquanto adaptávamos o algoritmo em conjunto.