

Relatório do Sprint C de Algoritmia Avançada

**Turma: 3DD Grupo 2**

**Membros do grupo:**

1181255 Nuno Teixeira

1181498 Rafael Barbarroxa

1181500 Vítor Crista

1181529 Diogo Sousa

**Docente/Orientador:**

António Silva (ASS)

**Unidade Curricular:**

Algoritmia Avançada



Índice:

[1. Introdução 3](#_Toc60690744)

[2. Representação do conhecimento do domínio 3](#_Toc60690745)

[3. Operadores de cruzamento e mutação implementados 4](#_Toc60690746)

[4. Aleatoriedade no cruzamento entre indivíduos da população 5](#_Toc60690747)

[5. Restrições consideradas (hard e soft constraints) e implementação do cálculo da função de avaliação 6](#_Toc60690748)

[6. Seleção da nova geração da população 8](#_Toc60690749)

[7. Parametrização da condição de término do AG 9](#_Toc60690750)

[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 12](#_Toc60690751)

[9. Proposta de duas heurísticas rápidas de criação de uma solução 13](#_Toc60690752)

[10. Conclusão 13](#_Toc60690753)

[Referências 14](#_Toc60690754)

Índice de Figuras:

[Figura 1- Gerar pontos de cruzamento 4](#_Toc60690758)

[Figura 2- Cruzar 4](#_Toc60690759)

[Figura 3- Método validar cruzamento 5](#_Toc60690760)

[Figura 4- Formar/criar tripletos 6](#_Toc60690761)

[Figura 5- Criar lista de tripletos de cada motorista 6](#_Toc60690762)

[Figura 6- Validação de almoço/jantar 7](#_Toc60690763)

[Figura 7- Validar tempo de descanso 8](#_Toc60690764)

[Figura 8- Validar horário de motorista 8](#_Toc60690765)

[Figura 9- Solução não puramente elitista 9](#_Toc60690766)

[Figura 10- Escolha das opções de término 10](#_Toc60690767)

[Figura 11- Validar opções de término 11](#_Toc60690768)

[Figura 12- Gráfico da evolução do valor da função 12](#_Toc60690769)

# Introdução

Durante a realização do Sprint C foi adaptado e analisado um algoritmo genético, com o objetivo de obter a melhor combinação de motoristas para realizar um determinado serviço.

Com isso em mente foi primeiramente necessário realizar o melhoramento do algoritmo genético fornecido para garantir que o melhor indivíduo da população corrente passa para a seguinte. Fazendo isso, evita-se que a sequência de cruzamento se dê do 1º e 2º elementos, 3º e 4º e assim sucessivamente, aplicando-se um método de seleção não puramente elitista, terminando com a criação de várias criações de término.

De seguida, foi feita uma análise profunda aos elementos de domínio que estão envolvidos na resolução do problema e a sua conversão para factos, podendo então, passar-se, depois de uma nova análise ao algoritmo fornecido, a uma adaptação do mesmo.

Finalmente, depois de adaptado o algoritmo genético, realizou-se uma análise da evolução do valor da função de avaliação do melhor indivíduo e da média de todos, representando o progresso das gerações ao longo do tempo e assim podendo-se verificar a qualidade do algoritmo.

# Representação do conhecimento do domínio

O domínio para este Sprint de ALGAV não corresponde na totalidade ao domínio de LAPR5, sendo apenas necessários os seguintes factos:

* **Horário:** definição horário de um conjunto de percursos.

Representado pelo facto **horario/6**, correspondente a horario (identificador de percurso, identificador de viagem, lista do tempo de passagem a cada nó do percurso).

* **Bloco de trabalho/Work Block**: conjunto de viagens feitas, sem interrupções, pela mesma viatura, e com ou sem interrupções, pelo mesmo tripulante.

Os blocos de trabalho podem agregar uma ou mais viagens e estão conectados aos horários, já que, o tempo de partida do bloco corresponde ao primeiro elemento da lista de passagem do horário menos 120s, e o tempo de término ao último (das viagens respetivas).

Assim sendo, o bloco de trabalho é representado em PROLOG por workblock (identificador de bloco de trabalho, lista de identificadores de viagens, tempo de partida, tempo de término).

* **Serviço de viatura/Vehicle Duty:** corresponde ao período de trabalho diário de uma viatura.

Representado por vehicleduty (identificador de serviço de viatura, lista de identificadores de bloco de trabalho), o serviço de viatura vai ser o meio com o qual se faz a ligação entre os tripulantes que o vão executar e os blocos de trabalho.

* **Lista de motoristas para o serviço de viatura:** não tem representação no domínio de LAPR5, mas é necessário para a resolução do problema, permitindo saber quantos workblocks vão ser executados por um determinado condutor para um serviço de viatura.

Corresponde ao facto **lista\_motoristas\_n\_workblocks** (identificador de serviço de viatura, lista de condutores e respetivos número de workblocks a executar).

* **Motorista/Driver:** criado no âmbito de averiguar quando é que o motorista entra e sai do trabalho, driver (identificador de driver, data de início do trabalho, data de fim do trabalho).

# Operadores de cruzamento e mutação implementados

O cruzamento é parte integral do algoritmo genético, pois é este que, juntamente com as mutações, é responsável pela diversidade das gerações e consequentemente pela obtenção do melhor resultado possível.

Assim sendo, o cruzamento e a mutação tiveram de ser adaptados para o problema dos motoristas, alterando-se o método que é usado em ambos para gerar os pontos de cruzamento, bem como condições para impedir que os motoristas excedam o número de blocos de trabalho que podem realizar por serviço.

O algoritmo realiza cruzamentos 2 a 2, isto é, 1º vai cruzar com o 2º, o 2º com o 3º e assim sucessivamente, utilizando os pontos de cruzamento, que são gerados ao dividir da dimensão da lista por 3, como divisores das partes que vão ser trocadas.

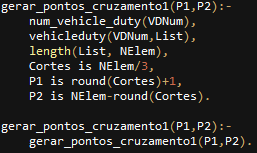


Figura 1- Gerar pontos de cruzamento

Conhecendo estas 3 partes, o algoritmo é responsável por criar uma nova população com o 3º terço da 2ª população, 2º terço da 1º população e 1º terço da 2ª população, nos seus 3º, 2º e 1º terços respetivamente.

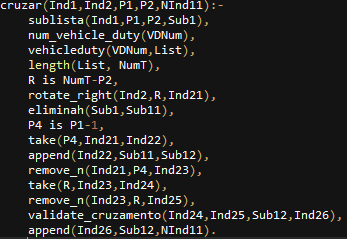


Figura 2- Cruzar

É de reforçar que apenas isso não garante um cruzamento correto, já que, quando o último terço é introduzido, este pode provocar problemas porque não há garantia que não existam motoristas a realizar blocos de trabalho a mais e outros a menos. Com isso em mente é necessário criar um método que na introdução do último terço (neste caso o 1º terço da 2º população), valide se só são introduzidos motoristas que ainda podem cumprir blocos de trabalho.

No caso de isso acontecer, vão ser introduzidos os motoristas que ainda têm blocos de trabalho disponíveis pela ordem em que aparecem no 2º terço da 2º população, sendo necessário criar métodos que realizem essa conversão. Isso é possível analisando quantos blocos de trabalho faltam para cada motorista, eliminando todos os que não podem fazer mais da lista do 2º terço da 2ª população para serem adicionados à lista final quando necessário.

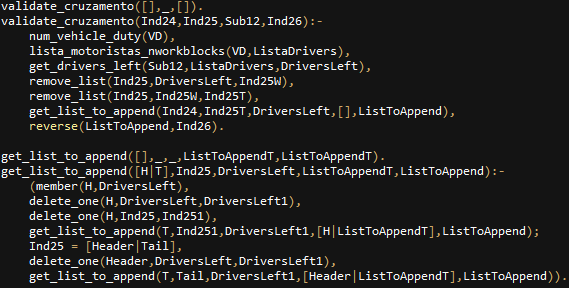


Figura 3- Método validar cruzamento

# Aleatoriedade no cruzamento entre indivíduos da população

Com o objetivo de evitar que a sequência de cruzamentos seja sempre entre elementos definidos dois a dois, o que pode levar à estabilização da população, permutou-se aleatoriamente a lista que contém os elementos da população. Este tipo de solução permite não modificar o método de cruzamento (continuando a cruzar dois a dois), mas como os elementos vão estar organizados aleatoriamente, deixa de ser um problema.

Para tal, utilizou-se o **random\_permutation/2** do SWI Prolog, correndo o método de cruzamento com a lista aleatória gerada.

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

Para o sucesso de um algoritmo genético é necessário a existência de regras de avaliação que penalizem os que não as atendam. Com isso em mente, ao adaptar o algoritmo para o problema dos motoristas, foi necessário definir um conjunto de hard e soft constraints e os seus respetivos pesos.

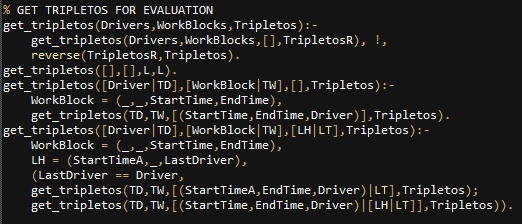
De modo a facilitar os cálculos das contraints, foram criadas agendas temporais para os motoristas. Primeiramente, criou-se uma lista de tripletos com os limites superiores e inferiores do intervalo de tempo do bloco de trabalho, depois, através dessa lista, os tripletos seguidos do mesmo motorista foram agregados.

Figura 4- Formar/criar tripletos

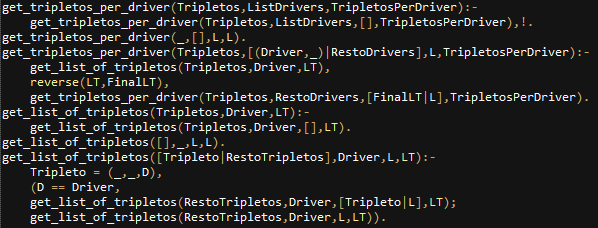
Após ter uma lista com todos os tripletos condensados, basta agora agrupar todos os tripletos de um motorista numa lista única dele, resultando assim uma lista com listas de tripletos de cada motorista.

Figura 5- Criar lista de tripletos de cada motorista

Posto isto foram escolhidas as seguintes hard constraints:

* Entre os instantes 39600 (11:00) e 54000 (15:00), assim como, 64800 (18:00) e 79200 (22:00), o motorista tem de ter pelo menos 3600s (1 hora, valor assumido) livres para almoçar e jantar, respetivamente.

Esta implementação é possível ao verificar se durante os períodos restritivos, o motorista tem mais de 3 horas de serviço, se for esse o caso, esse tempo será multiplicado por 8 e somado às restantes constraints.

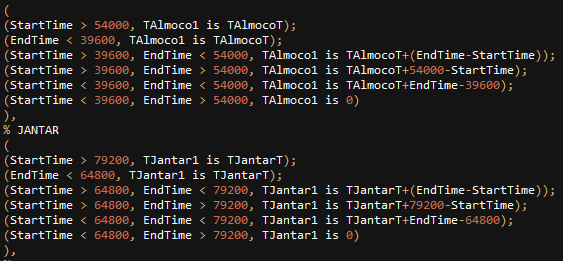


Figura 6- Validação de almoço/jantar

* O motorista só pode conduzir até 14400s (4 horas) consecutivas.

Isto é calculável utilizando os tripletos do motorista em causa, se em algum deles a diferença entre a hora final e a hora inicial for superior a 14400s, então somar-se-à ao valor das constraints 14400-(valor resultante da diferença) multiplicado por 10.

* O motorista só pode conduzir até 28800s (8 horas) diariamente.

Este valor é atingido ao fazer o somatório dos tempos de condução (diferença entre a hora inicial e final) de todos os tripletos. Caso esse valor seja superior a 28800s, utilizamos o mesmo método de calcular o tempo que o motorista conduziu a mais e multiplicamo-lo por 10.

* Se o motorista atingir 14400s consecutivos a conduzir, tem de fazer um descanso de 3600s.

Antes de calcular o descanso é necessário saber se no tripleto anterior o motorista conduziu 4 ou mais horas, caso seja verdade, vamos atribuir o tempo final da viagem a uma variável, caso contrário a variável terá o valor zero. Isto permite que no tripleto seguinte se faça a diferença entre o tempo de início com o guardado nessa variável, e se o valor for inferior a 3600, multiplicasse o tempo que ele não descansou, 3600-(tempo de descanso) e multiplicasse esse valor por 10 para ser somado ao total.

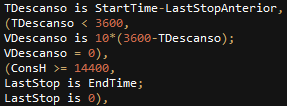


Figura 7- Validar tempo de descanso

* O motorista tem um determinado horário que tem de ser seguido.

Esta validação é obtida percorrendo os tripletos de um determinado motorista e verificando se algum valor neles presentes está fora do horário de trabalho do motorista. O Horário de trabalho é representado no facto **driver** que contém a hora de entrada, bem como a de saída do trabalho. Este cálculo tem de ser feito para todos os tripletos e incrementado, para que no final seja multiplicado por 8 e adicionado ao total de constraints.

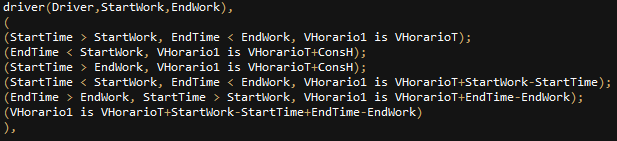


Figura 8- Validar horário de motorista

Neste projeto não foram implementados quaisquer soft constraints, mas uma potencial seria no caso de os motoristas terem um horário preferencial onde gostariam de trabalhar e caso o valor dos tripletos estivesse fora desse horário, o valor da diferença seria somado às constraints (multiplicador de 1 por ser uma soft constraint e não ter tanto peso na avaliação).

# Seleção da nova geração da população

Com o propósito de diversificar o gerador de gerações, adaptando-o para não ser puramente elitista, criou-se um algoritmo que escolhe os indivíduos que passam para a geração seguinte com base nos da atual e nos resultados dos cruzamentos e mutações. Para além disso, também é importante guardar o melhor indivíduo da geração atual, já que, devia ao resultado dos cruzamentos e mutações este pode ser perdido e nunca mais se encontrar uma solução tão boa.

Tendo isto em mente, no início do processo a melhor solução da geração atual é guardada para ser adicionada à nova posteriormente, sendo gerada uma nova lista de indivíduos através dos cruzamentos e mutações, lista essa que se vai juntar à da geração atual eliminando quaisquer repetidos. Após ordenada, a lista terá todos os elementos únicos da geração atual e da resultante dos cruzamentos, permitindo então passar para a escolha dos que passam para a geração nova. De maneira a não se ter uma seleção elitista, apenas são selecionados os primeiros 30% de N (número de indivíduos na população) através da avaliação, neste caso os primeiro N\*0.3 elementos da lista ordenada. Para os restantes indivíduos, a sua avaliação vai ser multiplicada por um número aleatório entre 1 e 0 e o resultado vai passar a ser a sua avaliação temporária, avaliação essa, que vai ser a fonte de escolha dos restantes elementos da geração.

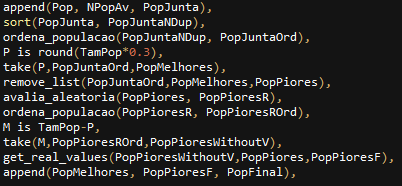
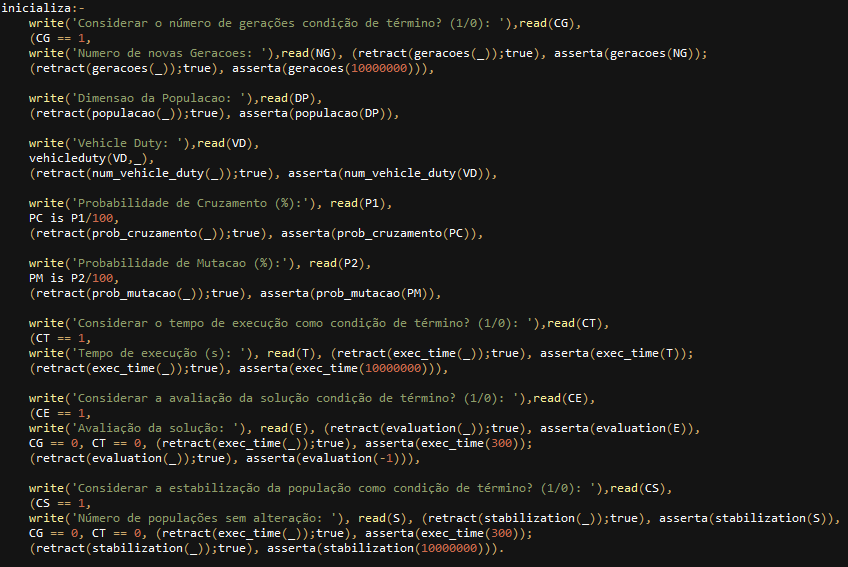


Figura 9- Solução não puramente elitista

Finalmente, caso não esteja já presente na lista formada, o que vai acontecer na maior parte das situações, tendo em conta que o melhor da lista atual genericamente faz parte dos melhores 30% da conjunta, o melhor da lista atual será adicionado no lugar do de pior avaliação.

# Parametrização da condição de término do AG

No início do programa é feita uma série de perguntas ao utilizador, com o intuito de conhecer o que é pretendido do algoritmo. Essas perguntas vão definir as condições de paragem do algoritmo, podendo este acabar porque o número de gerações, a avaliação, o tempo ou a estabilização da solução foram atingidos. O utilizador pode escolher todas as opções, terminando quando a primeira se verificar ou nenhuma, o que faz com que o algoritmo corra durante 120s (5 minutos), valores definidos para evitar a criação de um loop infinito.

 Figura 10- Escolha das opções de término

No caso da única opção ser a avaliação da população ou a estabilização das soluções, também é definido automaticamente um periodo de 120s para o algoritmo correr, já que, a avaliação escolhida pode ser menor que o minorante, o que causaria um loop infinito. Outra coisa que poderia acontecer seria a solução nunca estabilizar, isto é, o número de soluções iguais seguidas que o utilizador define pode nunca acontecer, tendo em conta que como o critério de seleção não é puramente elitista, dificultaria a estabilização.

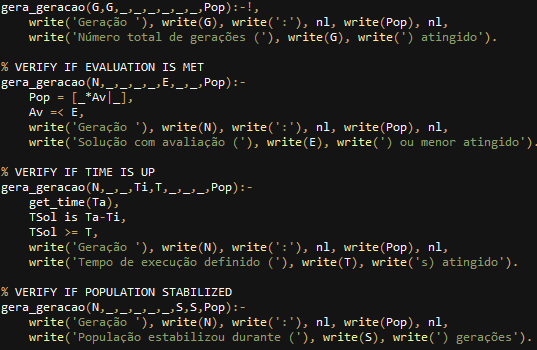


Figura 11- Validar opções de término

Como está implícito, o algoritmo incrementa o número de gerações e termina quando o valor definido é atingido, no caso da avaliação, esta é validada no início de todas as gerações até ser igual ou menor que a pretendida. Em relação ao tempo, este é calculado inicializando uma variável com o tempo no momento de início do processo, e quando a diferença do tempo atual para o de início for igual ou superior ao definido (pelo utilizador ou pelo programa) termina o processo. Para saber se a solução estabiliza sempre que uma geração é igual à anterior é incrementada uma variável, que depois é comparada ao número definido para terminar o programa.

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

Como algoritmo genético, quantas mais gerações existirem melhor será o resultado final, já que não só há mais oportunidades para cruzamentos e mutação, como também as gerações foram avaliadas mais vezes, e como tal os melhores indivíduos foram-se mantendo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nº Gerações | Prob de cruzamento | Prob de mutação | Melhor resultado | Média |
| 5 | 50 | 25 | 141,120 | 190,640 |
| 10 | 50 | 25 | 132,000 | 179,840 |
| 20 | 50 | 25 | 129,120 | 163,760 |
| 40 | 50 | 25 | 102,240 | 142,480 |
| 80 | 50 | 25 | 100,320 | 133,440 |
| 120 | 50 | 25 | 99,360 | 128,426 |
| 200 | 50 | 25 | 97,440 | 122,960 |
| 400 | 50 | 25 | 87,360 | 114,880 |
| 1000 | 50 | 25 | 73,440 | 92,840 |

Tabela 1- Evolução do valor da função

Figura 12- Gráfico da evolução do valor da função

O gráfico gerado demonstra exatamente isso, resultando uma diminuição da avaliação com o aumento do nº de gerações.

Apesar de isto se verificar, era também possível que com um número reduzido de gerações se gerasse um melhor resultado que com um número maior, porque como os cruzamentos e mutações são aleatórios, podiam existir as condições necessárias de imediato para gerar a melhor geração, ainda que fosse improvável.

# Proposta de duas heurísticas rápidas de criação de uma solução

Um possível melhoramento do algoritmo seria a introdução de heurísticas rápidas que seriam responsáveis por criar as primeiras populações. Por exemplo, utilizar uma heurística que organiza os motoristas sequencialmente, por ordem de horário, sem exceder as 4 horas seguidas, de maneira a evitar a hard constraint. Outra opção seria usar uma heurística que em vez de utilizar o horário contratual dos motoristas, usasse a sua preferência horária, evitando que tanto a hard como a soft constraints, permitissem que as gerações que se seguem começassem num ponto mais avançado, onde algumas constraint já foram salvaguardadas, o que concedia que o algoritmo atingisse o minorante sem necessitar de tantas gerações.

# Conclusão

No que toca ao cruzamento, foi optado por manter o núcleo do método intacto, tendo sido apenas modificado a maneira como o mesmo validava as restrições do número de blocos de trabalho que cada motorista poderia realizar por serviço. Desta maneira, de modo a evitar o estagnamento dos cruzamentos, permutou-se as gerações, facilitando assim o processo.

Em relação à avaliação, a introdução de hard constraints com diferentes pesos que dependem da severidade dos casos, tornou o algoritmo capaz de facilmente diferenciar a qualidade das populações geradas. Escolheu-se dar um multiplicador de 10, ou seja, maior consideração a tudo o que ponha em causa a saúde física do motorista, como todos os transportados no autocarro, neste caso, horas seguidas a conduzir e descanso obrigatório, assim como, restrições contratuais como horas de trabalho diário. As restantes hard constraints, nomeadamente refeições e horários de motoristas têm multiplicador de 8, diferenciando-as das outras, deixando ainda espaço para outras que possam vir a ser introduzidas.

A análise da evolução da avaliação demonstrou que a tendência da mesma é diminuir com o aumento do número de gerações, apesar de o exemplo selecionado ter sido “feliz”, já que, foram corridos outros exemplos onde o desenvolvimento da avaliação não era tão linear devido à aleatoriedade dos cruzamentos e mutações.

Concluindo, o algoritmo genético mostrou-se capaz de gerar soluções razoáveis para o problema, dado o número limitado de gerações, demonstrando a qualidade do cruzamento de populações, bem como o avaliador.

**Distribuição de tarefas**:

Diogo Sousa: Operadores de cruzamento e mutação.  
Nuno Teixeira: Implementação do cálculo da função de avaliação, análise da evolução da função de  
avaliação.  
Rafael Barbarroxa: Parametrização da condição de término do AG.  
Vítor Crista: Aleatoriedade no cruzamento entre população, seleção da nova geração da população.

# Referências

(s.d.). *Algoritmos Genéticos Unidade Curricular: Algoritmia Avançada Curso: Licenciatura em Engenharia Informática (ISEP).*

Curricular, U., & Avançada, A. (s.d.). *Adequação do Algoritmo Genético base fornecido para o Sequenciamento de Motoristas num Autocarro.*

(s.d.). *Entrega para o Sprint C de ALGAV.*