Um Algoritmo Genético Para a Resolução do Problema de Tabelamento de Horários, Professores e Disciplinas Para a Universidade Estadual do Piauí

Jefferson Carvalho[1], **Francisco Júnior**[1]

[1]Universidade Estadual do Piauí (UESPI)
Parnaíba – PI – Brasil

Abstract. *This work presents a genetic algorithm capable of solving the timetabling problemem for Piaui's state university(Universidade Estadual do Piauí). It uses the single point crossover technique and two mutation operators in order to create new generations, as well as a genetic repair operator for infeasible solutions.*

Resumo. *Este trabalho apresenta um algoritmo genético capaz de resolver o problema de tabelamento de horários para a Universidade Estadual do Piauí. Ele se utiliza da técnica de crossover de ponto único e dois operadores de mutação para criar novas gerações, além de um operador de reparo genético para soluções inviáveis.*

1. Introdução

Semestralmente, coordenações de diversos cursos, sejam eles de ensino médio ou superior, se deparam com o problema de alocação de horários de professores e disciplinas.

Deseja-se realizar essa tarefa encontrando uma configuração de horários livre de conflitos e que obedeça à certas restrições. É muito comum nas instituições de ensino brasileiras que esse árduo processo seja realizado de maneira manual. Isso faz com que o processo possa levar dias, e até semanas. Uma maneira computacional de se realizar esse processo é desejável, tendo em vista que o tempo gasto nessa tarefa poderia ser melhor empregado pelo responsável por ela.

Os problemas de tabelamento de horário, ou *Timetabling Problems*, são classificados na Ciência da Computação como problemas de classe NP-completo ou NP-difícil[Poulsen 2012]. Essas classificações indicam que não são conhecidos na literatura algoritmos de complexidade polinomial que encontrem uma solução para este tipo de problema. Isso faz com que seja necessário que se recorra à soluções heurísticas para que se convirja a uma solução satisfatória em um intervalo de tempo que seja viável. Dentre as várias heurísticas e meta-heurísticas utilizadas para esse fim, encontram-se *simulated annealing*, busca tabu, redes neurais e algoritmos genéticos[Raghavjee and Pillay 2010]. Este trabalho terá como foco os algoritmos genéticos.

Segundo [Sousa et al. 2008], o problema de tabelamento de horários escolares, ou *school timetabling problem*(STP) possui inúmeras variações devido ao fato de que países, regiões dentro de um país ou até instituições de ensino de uma mesma região adotam critérios educacionais diferentes entre si. Sendo assim, esse problema é de difícil generalização, e cada variação desse problema precisa de uma implementação específica da heurística desejada a fim de se obter resultados que melhor se adéquem às restrições.

Este trabalho terá como foco a Universidade Estadual do Piauí(UESPI), que se classifica como STP pois segue o regime do seriado.

2. Trabalhos Relacionados

Entre outros trabalhos que tratam problemas parecidos e que também utilizam algoritmos genéticos, estão [Raghavjee and Pillay 2010], que utiliza um algoritmo genético em duas fases, onde a primeira é responsável por criar soluções que respeitam as restrições fortes do problema, e a segunda fase é responsável pelas restrições fracas; [Nunes et al. 2013], que assim como neste trabalho, utiliza um operador de reparo genético para que todas as soluções sejam factíveis; E por fim, [Aragão 2010] resolve o problema também para a UESPI, porém, sem resolver a relação entre professores e disciplinas (essa relação é constante e alheia ao algoritmo) e sem utilizar as técnicas de reparo genético e de criação de novas gerações abordadas neste trabalho.

3. Referencial Teórico

3.1. Problemas de Otimização

De acordo com [Cormen et al. 2009] problemas de otimização são problemas que possuem múltiplas soluções, onde cada solução possui um valor associado a ela que determina a sua qualidade, e o objetivo é encontrar um valor ótimo(mínimo ou máximo). A essa solução, damos o nome de *solução ótima*. É possível que existam várias soluções ótimas para um mesmo problema.

A grande maioria dos problemas de otimização estudados na ciência da computação são problemas da classe NP-completo ou NP-difícil. Problemas dessas classes não possuem soluções de complexidade polinomial conhecidas, exceto se $P = NP$ [Cormen et al. 2009].

Tendo isso em vista, algoritmos que garantem a melhor resposta para esse tipo de problema muitas vezes são inviáveis de serem aplicados devido a sua alta complexidade de tempo. Porém, nem sempre é necessário que se chegue na *melhor* resposta. Na prática, uma resposta considerada "boa" já é suficiente. Para isso, são utilizadas heurísticas ou meta-heurísticas para se chegar a uma solução. Heurísticas são algoritmos que fornecem soluções aproximadas para problemas específicos. Meta-heurísticas funcionam da mesma forma, porém, podem ser adaptadas para qualquer problema de otimização com relativa facilidade [Rothlauf 2011].

Como existem múltiplas soluções, uma maneira de se interpretar o funcionamento de algoritmos que resolvem esse tipo de problema é imaginá-los como algoritmos de busca. Dentro de um espaço de soluções, procura-se a melhor dentre todas elas, e essa solução recebe o nome de solução *ótima global*. Porém, nem sempre se chega a essa solução, embora esse seja o objetivo. É possível que o algoritmo acabe ficando preso em uma *vizinhança*, isto é, um conjunto de soluções que são consideradas próximas de acordo com critérios de implementação, e encontre apenas a melhor solução para aquela região do espaço de busca. A ela, damos o nome de solução *ótima local*. Um *movimento* é quando o algoritmo passa de uma solução para outra dentro do espaço de busca [Rothlauf 2011].

Um exemplo clássico desse tipo de problema é o *problema do caixeiro viajante*, ou *travelling salesman problem*(TSP). Nesse problema, dado um conjunto de cidades e

estradas que ligam essas cidades, deve-se encontrar o menor caminho possível que passe por todas as cidades sem repetir nenhuma, com exceção da última cidade a ser visitada, que deve ser a primeira. Trata-se de um problema de otimização, onde se deve minimizar o custo da solução, que é a distância total a ser percorrida. Problemas de tabelamento de horários também são classificados como problemas de otimização.

3.2. *School Timetabling Problem*(STP)

O problema de tabelamento de horários escolares, ou *School Timetabling Problem* é uma categoria de problema de tabelamento. Problemas de tabelamento são problemas que consistem em alocar um número de eventos a uma quantidade finita de intervalos de tempo, ou períodos, de forma que as restrições necessárias sejam satisfeitas[Burke and Newall 1999].

O STP é uma especialização desse problema, onde os eventos a serem alocados são *aulas*. Uma aula será definida como uma *tupla* que reúne um *professor*, uma *disciplina* e uma *turma*. Algumas formulações do problema, como em [Abramson and Abela 1991], incluem também uma *sala* na tupla. O problema consiste em alocar aulas a intervalos de tempo pré determinados, ou *períodos*. Cada período é um intervalo de tempo na semana onde uma aula pode ser ministrada. Assume-se aqui que todas as aulas terão a mesma duração. A quantidade de períodos a serem considerados será a quantidade de períodos disponíveis em uma semana, visto que as grades de horário tradicionalmente são organizadas semanalmente. As restrições do problema são arbitrárias. Exemplos clássicos de restrições são que um professor ou uma turma não podem ter mais de uma aula em um mesmo período, e que as cargas horárias tanto dos professores quanto das disciplinas devem ser respeitadas. Uma solução pode ser vista como um vetor de períodos, onde cada período possui zero ou mais tuplas associadas a ele. Essa solução pode facilmente ser convertida em matrizes de horários para cada turma para uma melhor visualização.

Para se medir a qualidade de uma solução, é possível definir uma função objetivo que fará um cálculo com base na quantidade e no tipo de restrições que são violadas[Abramson and Abela 1991]. Cada solução terá um custo associado a ela. O custo ideal de uma solução é zero. Cada restrição violada aumentará esse custo e o quanto ele será aumentado pode depender do tipo de restrições violadas. Logo, o STP pode ser visto como um problema de otimização, onde o objetivo da heurística empregada será minimizar o valor do custo da solução a ser encontrada.

3.3. Algoritmos Genéticos

Segundo [Russell and Norvig 2016], algoritmos genéticos são algoritmos de inteligência artificial inspirados no processo de evolução das espécies, de acordo com a teoria da evolução de Darwin. A teoria da evolução diz que, na natureza, aqueles indivíduos que melhor se adaptam ao ambiente são os que sobrevivem, enquanto que os que não se adaptam tendem a morrer prematuramente. Os indivíduos que se adaptaram repassam as suas características, ou genes, para as próximas gerações de indivíduos através de processos de reprodução. Dessa forma, quanto mais gerações, mais adaptados ao ambiente os indivíduos estão, pois aqueles com características consideradas “ruins” desaparecem, enquanto que aqueles com características “boas” sobrevivem e elas tendem a ser transmitidas para as gerações futuras.

Os algoritmos genéticos seguem essa lógica, e são muito utilizados para a resolução de problemas de otimização, incluindo o STP. Eles começam com um conjunto de soluções iniciais para o problema, a *população*. Cada solução é conhecida como *cromossomo*. Depois disso, começa um ciclo de *seleção*, *reprodução* ou *crossover* e *mutação*. Isso é feito por um certo número de vezes. Para cada vez, ou iteração, em que esse processo é repedido dá-se o nome de *geração*. É uma decisão do implementador do algoritmo se ele terá um número fixo de gerações, ou se ele parará de acordo com algum critério arbitrário. Ao final de cada geração, a nova população gerada através do *crossover* é substituída pela anterior. Alguns cromossomos da população anterior podem ser preservados. Geralmente são preservados aqueles que melhor se adaptaram, em um processo conhecido como *elitismo*.

O grau de adaptação de um cromossomo é calculado através de uma função *fitness*. O cálculo do *fitness* varia de acordo com o problema. Esse valor é muito importante no algoritmo, pois além de medir a qualidade da solução de cada cromossomo(o cálculo é realizado para cada um deles) ele também é utilizado no processo de seleção.

A seleção visa escolher quais são os cromossomos que participarão do *crossover*. Existem vários métodos para se fazer isso, mas os mais conhecidos são o *método da roleta* e o *método do torneio*. Ambos escolhem dois cromossomos, e a partir deles, a nova população é gerada. O método da roleta escolhe os "pais" de maneira probabilística. Quanto maior o *fitness* do cromossomo, maiores as chances dele ser selecionado. Já o método do torneio escolhe aleatoriamente uma determinada quantidade de cromossomos da população. Dentre os selecionados, escolhe-se o que tiver o maior *finess* entre eles. Esse processo é repetido para cada um dos pais.

O *crossover* combina os genes ou características dos pais escolhidos na seleção. O que será um gene depende de como o cromossomo será implementado, e isso pode interferir diretamente em como esse processo será realizado. Todos os métodos consistem em combinar os genes dos pais para gerar os filhos. Esse processo é repetido até que a quantidade de filhos necessária seja gerada.

Sempre após um filho ser gerado no *crossover*, verifica-se se algum gene dele será alterado. Isso acontece também de maneira probabilística, sendo que probabilidade de geralmente é muito baixa. A esse processo se dá o nome de *mutação*. Assim como o *crossover*, a maneira que a mutação ocorre também depende da maneira com que o cromossomo foi implementado. A mutação é extremamente importante para evitar que o algoritmo convirja para uma solução ótima local, pois ela pode fazer com que o algoritmo vá para um novo espaço de busca, aumentando assim as chances para que se chegue a um ótimo global.

4. Problematização

4.1. Definição do problema

O STP apresentado neste trabalho é adaptado para a realidade da Universidade Estadual do Piauí(UESPI). Semestralmente, coordenadores dos cursos da universidade se deparam com o problema de criar uma grade horária obedecendo um determinado conjunto de restrições. O problema considera apenas um curso individualmente, fazendo com que a solução tenha que ser aplicada para cada curso de forma separada.

Percebe-se que a informação a respeito de toda a grade horária do semestre de um curso pode ser resumida em tabelas semanais de aulas para cada turma. Sendo assim, organizando-se tabelas semanais, organizam-se as aulas para o semestre inteiro.

A princípio, existe um conjunto de disciplinas para serem ofertadas em um semestre. Essas disciplinas podem ser relacionadas aos blocos em que elas devem ser ministradas, dividindo-as em grupos. Existirá uma tabela, ou grade horária para cada um desses grupos, ou seja, uma grade horária por turma naquele curso.

Cada disciplina possui uma carga horária. Na UESPI, essas cargas horárias podem ser de 30, 60 ou 120 horas. Uma disciplina de 30 horas possui uma aula semanal, uma disciplina de 60 horas possui duas aulas semanais e uma de 120 horas possui três aulas semanais. A carga horária das disciplinas têm que ser cumprida e não pode ser excedida.

Um conjunto de professores alocados ao curso estão aptos para ministrar essas disciplinas. É preciso relacionar um professor para cada uma delas. Cada professor possui preferências por cada disciplina, que devem ser levadas em consideração no processo de construção das grades horárias.

Os professores também possuem cargas horárias máximas. Esse valor pode variar de professor para professor dependendo do seu vínculo empregatício com a instituição e de projetos de pesquisa e extensão. Essas informações, porém, não são essenciais para o problema: é necessário saber apenas qual a carga horária aquele professor possui disponível para ministrar aulas.

Os professores irão ministrar as aulas para as disciplinas selecionadas em determinados períodos da semana. Cada período corresponde ao horário de uma aula. Assim como a carga horária das disciplinas pode ser expressa em quantidade de aulas semanais, ou número de períodos semanais, a carga horária máxima de um professor também pode ser expressa pela quantidade de períodos que ele possui disponível em uma semana. Um professor também possui períodos em que ele não pode ministrar aula. Isso foi incluído no problema para casos de professores que não são dedicação exclusiva e ministram aulas em outras instituições de ensino.

Uma aula é uma tupla que relaciona um professor, uma disciplina e uma turma. As aulas devem ser alocadas em períodos da semana. Deve haver uma quantidade de aulas suficiente para cumprir a carga horária de todas as disciplinas, e deve-se evitar exceder a carga horária máxima dos professores. É interessante também evitar janelas nas grades de horário e evitar que existam aulas em múltiplos turnos de um mesmo dia para uma mesma turma.

Sendo assim, o problema consiste em encontrar uma combinação ótima de professores e disciplinas, a partir da qual serão alocadas aulas em diferentes períodos da semana com o intuito de cumprir a carga horária de todas as disciplinas. A combinação será considerada ótima caso ela não viole nenhuma restrição. Existem restrições fortes e fracas. As restrições fortes não podem ser violadas, pois elas devem ser obedecidas para que a solução seja válida. Exemplos são um professor não ministrar mais de uma aula em um mesmo período, uma turma não ter mais de uma aula em um mesmo período, cada disciplina deve ter somente um único professor, etc. As restrições fracas podem ser violadas, mas é desejável que isso não aconteça. A qualidade de uma solução válida é determinada a partir da quantidade de restrições fracas que ela viola. Exemplos de restrições fracas são

não haver janelas nos horários, a preferência dos professores pelas disciplinas deve ser a maior possível, não se deve alocar aulas em períodos não ideais para um professor, entre outras.

4.2. Definição formal do problema

De maneira mais formal, o problema tratado neste artigo pode ser definido da seguinte forma:

- Seja um conjunto de professores $T = \{t_1, ..., t_m\}$
- Seja um conjunto de disciplinas $S = \{s_1, ..., s_n\}$
- Seja um conjunto de turmas $C = \{c_1, ..., c_k\}$
- Cada disciplina está associada a uma turma
- Seja d a quantidade de dias letivos da semana e p a quantidade de períodos disponíveis por dia, tal que em cada período possa ser ministrada uma aula
- Seja um conjunto de preferências de professores por disciplinas definidos por uma matriz bidimensional P de dimensões $m \times n$ tal que $P_{ij} \in [1, 5]$ onde as linhas representam os professores e as colunas representam disciplinas, e cada valor nessa matriz representa a preferência que um professor possui por uma disciplina
- Seja um conjunto de períodos disponíveis definidos por uma matriz tridimensional M de dimensões $d \times p \times k$ onde as linhas representam dias da semana, colunas representam períodos e cada dimensão representa uma turma de um curso
- Seja um conjunto de períodos não disponíveis para professores representado por uma matriz tridimensional N de dimensões $d \times p \times m$ tal que $N_{ijk} \in \{0, 1\}$ onde as linhas representam períodos, colunas representam dias da semana e cada dimensão representa um professor e que elementos de valor 1 nessa matriz representam períodos que o professor t não pode ministrar aulas
- Seja uma aula definida pelo vetor $L = (s, t, i, j)$ onde
  - $s \in S$ é a disciplina da aula
  - $t \in T$ é o professor responsável pela disciplina
  - $i \in \mathbb{N}^* \mid i \leq h$ é o período da aula
  - $j \in \mathbb{N}^* \mid j \leq d$ é o dia da semana
- Seja um conjunto com a quantidade máxima de aulas semanais para cada professor $TW = \{tw_1, ...tw_m\}$, sendo que tw_i é a quantidade máxima de aulas do professor t_i
- Seja um conjunto com a quantidade máxima de aulas semanais para cada disciplina $SW = \{sw_1, ..., sw_n\}$ sendo que sw_i é a quantidade máxima de aulas da disciplina s_i

Deve-se criar um conjunto A de aulas e alocá-las à diferentes posições da matriz M, tal que as seguintes restrições fortes sejam obrigatoriamente satisfeitas:

- A carga horária da disciplina deve ser estritamente respeitada
- Um professor não pode ministrar mais de uma aula em um mesmo período
- Uma turma não pode ter mais de uma aula em um mesmo período
- Cada disciplina deve ser ministrada por apenas um único professor

As seguintes restrições fracas não são obrigatórias, mas elas devem ser satisfeitas sempre que possível para que se tenha uma solução de melhor qualidade:

- Não podem haver mais de uma aula de uma mesma disciplina em um mesmo dia

- Não podem haver janelas na grada horária, isto é, horários vago entre aulas
- Professores não devem ministrar aulas em períodos não disponíveis para eles
- Professores devem ministrar aquelas disciplinas por qual sua preferência é maior
- A carga horária máxima dos professores deve ser respeitada
- Aulas para uma turma devem sem ministradas em um mesmo turno do dia
- Um dia não pode ter apenas uma única aula para uma turma

5. Metodologia

5.1. Visão geral do algoritmo

O algoritmo genético proposto neste artigo seguirá uma estrutura parecida com a tradicional de um algoritmo genético, onde primeiramente será gerada a população inicial de cromossomos, calculado o *fitness* de cada solução gerada e começará um ciclo de criação de novas gerações, constituído por seleção, *crossover*, mutação e cálculo do *fitness* das soluções geradas. Isso vai acontecer até que um critério de parada seja satisfeito.

Entretanto, ele apresenta algumas diferenças. Conforme será comentado a seguir, nem toda a população será gerada através no *crossover*, mas uma parte dela será gerada diretamente por mutações. Em seguida, a nova solução será reparada caso ela viole alguma restrição forte, o que a invalidaria. Além disso, novas soluções serão incorporadas ao algoritmo somente se o *fitness* dela for maior que o *fitness* da solução representada pela iteração atual do processo de criação da nova população. Cada etapa desse processo será descrita a seguir.

Algorithm 1 Algoritmo genético

```
gerar população inicial
calcular o fitness das soluções geradas
while critério de parada não satisfeito do
    seleção
    for cada solução do
        crossover ou mutação
        reparos de restrições fortes violadas
        cálculo do fitness da nova solução
        if fitness da nova solução for maior que o fitness da solução atual then
            inserir nova solução na nova população
        else
            inserir solução atual na nova população
        end if
    end for
end while
```

5.2. Representação de um cromossomo

Uma solução consiste em um conjunto de tuplas, ou aulas, alocadas em períodos. Conforme dito anteriormente, cada período representa um intervalo de tempo onde uma tupla pode ser ministrada. Para se representar os períodos, foi utilizado um vetor cujo tamanho é igual a quantidade total de períodos em uma semana. Sabendo-se a quantidade de períodos por dia e a quantidade de dias letivos por semana, é possível mapear cada posição

do vetor a um determinado período da tabela de horários. Considerando uma semana com 5 dias letivos(segunda à sexta) e 6 períodos por dia(3 de manhã e 3 durante a tarde), as posições 1 à 6 do vetor corresponderiam as períodos da segunda-feira, as posições 7 à 12 aos períodos da terça-feira, e assim por diante.

Cada período pode possuir zero, uma ou mais tuplas, obedecendo as restrições fortes já mencionadas, de modo que não existirão mais de uma aula em um mesmo período com a mesma turma e/ou com o mesmo professor. Uma tupla irá simplesmente reunir um professor, uma disciplina e uma turma. Vale lembrar também, que cada solução possui a sua própria relação entre professores e disciplinas.

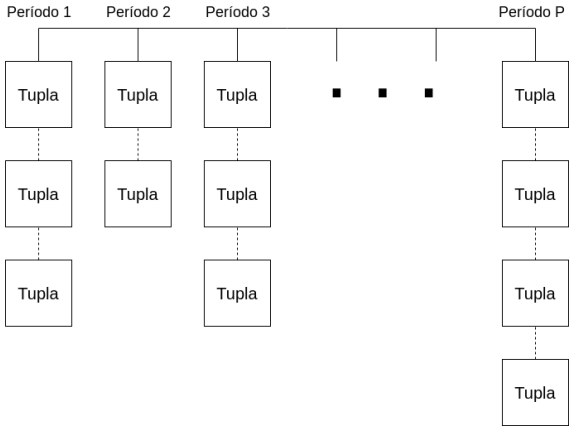

Figura 1. Representação de um vetor onde cada posição possui uma lista de tuplas

5.3. Função objetivo(*fitness*)

Cada solução possui um custo, que vai aumentando para cada restrição fraca violada. Ao se calcular o *fitness* da solução, começa-se com um custo inicial igual a 0, então de uma a uma, as restrições são avaliadas. Cada restrição violada aumenta o custo seguindo a relação demonstrada na tabela 1.

Tabela 1. Acréscimo de custo para cada restrição fraca violada

Aula de uma mesma disciplina em um mesmo dia	10 por aula extra
Período vazio entre duas aulas(janela)	30 por período
Professor lecionando em um período não disponível para ele	10 por aula no período
Preferência do professor por disciplina	*Ver tabela 2*
Professor com carga horária máxima excedida	1 por aula extra
Aulas para uma mesma turma em um mesmo dia em turnos diferentes	20 por aula
Turma com uma única aula em um dia	10

Dessa forma, o custo total de uma solução é o somatório dos acréscimos de todas as restrições violadas. A partir do custo, pode-se definir o *fitness* da solução. Por questões de conveniência para a implementação, definiu-se que quanto maior o *fitness*, melhor a solução. Para que um custo menor implique em um *fitness* maior, pode-se estabelecer um valor teto para o *fitness* e subtrair o custo desse valor.

Tabela 2. Acréscimo de custo de acordo com a preferência do professor por cada disciplina escolhida para ele

Valor da preferência	Acréscimo
5	0
4	2
3	4
2	6
1	30

5.4. População inicial

Para a primeira geração do algoritmo, escolhe-se aleatoriamente um professor para cada disciplina presente na grade de horários. Após isso, são geradas tuplas, ou aulas, a partir das escolhas feitas. Como cada disciplina está vinculada a uma turma, essa escolha também determina qual turma estará presente na tupla. Cada tupla gerada é colocada em uma posição aleatória do vetor de períodos.

Todo esse processo é feito observando as restrições fortes do problema, de modo que sempre que for escolhido um período onde o professor da tupla em questão já está ministrando uma aula, ou que a turma em questão já tem uma aula para assistir, o algoritmo escolhe uma outra posição para a tupla e a avalia novamente. Isso é feito até que seja gerado um número de tuplas suficiente para fechar a carga horária de todas as disciplinas.

5.5. Seleção

Para selecionar as soluções que seráo utilizadas na criação de novos indivíduos, foi utilizado o *Método da roleta*.

Neste método, quanto maior o *fitness* da solução, maior a probabilidade dela ser escolhida para ser utilizada no processo de criação da nova geração. O método é assim chamado porque lembra como uma roleta funciona na vida real. Quanto maior o *fitness*, maior a fatia que aquela solução terá na roleta, o que faz com que sejam maiores as chances dela ser escolhida.

A implementação deste método é simples, e segue o seguinte raciocínio: primeiro, realiza-se o somatório do *fitness* de todas as soluções da população; Após isso, escolhe-se um valor aleatório entre 1 e o total do somatório. Em seguida, percorre-se o vetor com os *fitness* das soluções, e para cada *fitness*, subtrai-se do valor aleatório escolhido o valor do *fitness* da solução da iteração atual. Esse processo continua até que o valor escolhido chegue a zero. No momento que isso ocorre, verifica-se a qual solução aquele fitness pertence. Aquela solução será então escolhida para o *crossover*.

Após isso, repete-se o processo de escolha de um número aleatório e subtrações sucessivas até que o resultado chegue a zero novamente. Neste momento, é escolhida a segunda solução que participará do *crossover*.

Como os soluções com os maiores *fitness* têm maior probabilidade de serem selecionadas, os genes que fazem com que essas soluções sejam boas tendem a serem transmitidos para a geração seguinte. Porém, como sempre existe a chance de uma solução com um *fitness* baixo ser selecionada, mantêm-se uma diversidade na população devido à variedade de genes, o que é útil para se explorar um espaço de busca maior.

5.6. Criação da nova geração

Em um algoritmo genético tradicional, após a seleção, utiliza-se um operador de *crossover* para se criar novos indivíduos ou soluções, e em seguida, os novos indivíduos passam pelo operador de mutação, que altera características desse novo indivíduo, geralmente com uma probabilidade muito pequena.

Neste trabalho, porém, usa-se uma ideia parecida com a utilizada por [Raghavjee and Pillay 2010], que modifica esse processo, fazendo com que o operador de mutação se torne um operador de reprodução. Em outras palavras, o operador de mutação é utilizado não para modificar uma solução que acabou de ser criada pelo *crossover*, mas sim para criar novas soluções a partir de soluções da geração anterior.

Entretanto, Raghavjee e Pillay utilizam *somente* operadores de mutação para se criar novos indivíduos. Neste trabalho, *crossover* não vai deixar de existir. Melhores resultados foram alcançados ao se alternar entre os operadores de mutação e *crossover* no processo de se criar uma nova geração.

Sendo assim, são criadas uma a uma as soluções que irão compor a nova geração de indivíduos. Para cada nova solução, o algoritmo decide probabilisticamente se vai utilizar *crossover* ou mutação. Uma vez decidido, o novo indivíduo será gerado a partir das soluções selecionadas na etapa anterior. Depois que ele é gerado, ele passará por um operador de reparo genético, assim como feito em [Nunes et al. 2013], caso seja detectado que restrições fortes estão sendo violadas. Após o reparo, seu *fitness* é calculado. Se ele for maior que o *fitness* da solução de mesmo índice da geração anterior, então a nova solução fará parte da próxima geração. Caso contrário, a nova solução é descartada, e a solução anterior será copiada para a nova geração. Sem esse artifício, boas soluções estavam sendo descartadas e o *fitness* das soluções não estava aumentando à medida que as gerações passavam.

5.7. Crossover

No *crossover*, se combinará as duas soluções escolhidas no processo de seleção para se criar uma nova solução. Para isso, características das soluções escolhidas serão combinadas, e a nova solução será um resultado dessa combinação.

O método de crossover utilizado foi o método de ponto simples, ou *single point crossover*. Este método consiste em aleatoriamente escolher um ponto de divisão nos cromossomos, ou soluções, que fazem parte do processo. Como trata-se de um único ponto, este ponto dividirá cada cromossomo em dois. A partir disto, cria-se um novo cromossomo, o *filho*. O filho é criado ao se combinar a primeira parte da primeira solução com a segunda parte da segunda solução.

Para este problema, cada solução foi definida como um vetor de períodos. Cada período pode ter até a aulas. O processo de *crossover* criará um novo vetor de períodos, combinando duas soluções diferentes. Isso pode fazer com que algumas restrições fortes do problema sejam violadas.

Como para cada solução é definida quais professores ministram quais disciplinas, é possível que uma mistura de soluções cause inconsistências na nova solução gerada, pois podem haver aulas de mesma disciplina com professores diferentes: isso ocorre porque é perfeitamente possível que um professor t_1 ministre uma determinada disciplina na

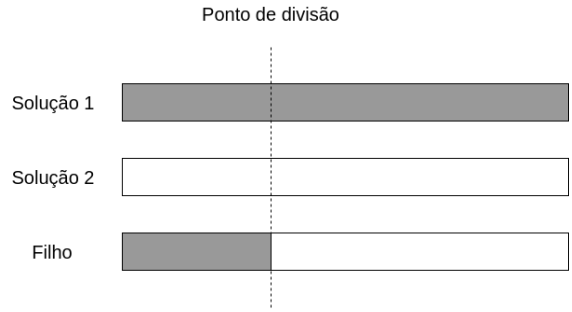

Figura 2. Crossover de ponto simples

primeira solução, mas na segunda solução essa disciplina tenha sido atribuída para um professor t_2. Tal inconsistência viola a restrição forte que que diz que cada disciplina deve ser ministrada por apenas um único professor. Para solucionar este problema, verifica-se onde existem os conflitos, ou seja, quais são os professores que estão ministrando a mesma disciplina, e então decide-se aleatoriamente para qual dos dois professores a disciplina será atribuída. Em seguida, removem-se as aulas onde a disciplina em questão é ministrada pelo o professor que não foi escolhido. A aleatoriedade foi utilizada com o intuito de se ajudar a manter a diversidade na população. Uma alternativa seria atribuir a disciplina para o professor que tiver a maior preferência por ela.

Uma outra restrição forte que pode ser violada neste processo é a da quantidade de aulas para cada disciplina. A combinação dos vetores de períodos pode fazer com que a solução gerada fique com aulas a menos ou com aulas a mais. Em todo caso, a carga horária da disciplina deixa de ser respeitada. Para as disciplinas em que a carga horária está aquém do necessário, criam-se novas aulas e elas são atribuídas a períodos aleatórios. Para as disciplinas em que a carga horária está além do necessário, percorrem-se os períodos do começo ao fim, e as aulas daquela disciplina que forem encontradas serão eliminadas, até que se atinja a carga horária da disciplina. Este problema é resolvido no reparo genético.

5.8. Mutação

Caso a nova solução seja gerada através de mutação, é preciso escolher aleatoriamente um dos dois operadores: o de *troca de aulas* e o de *troca de professores*. Existe uma chance igual para cada operador ser escolhido.

Se o operador de troca de aulas for escolhido, primeiro escolhe-se aleatoriamente dois períodos. Em seguida, verifica-se se nenhum dos dois está vazio, ou seja, se não existem aulas alocadas em nenhum deles. Caso ambos os períodos possuam aulas, escolhe-se também de forma aleatória uma aula de cada período, e então elas são trocadas de posição. Se um dos dois períodos escolhidos for vazio, simplesmente escolhe-se uma aula do outro período e ela é transferida para o que estiver desocupado. Nada é feito caso ambos os períodos escolhidos não possuam aulas associadas a eles.

O operador de troca de professores escolhe aleatoriamente duas disciplinas e troca os professores que as ministram.

5.9. Reparo

O reparo genético é feito caso a solução viole restrições fortes.

Primeiro, procura-se por aulas que estejam faltando para cada disciplina. Se estiverem faltando aulas, novas aulas são geradas e inseridas em períodos aleatórios até que a carga horária daquela disciplina seja cumprida.

Em seguida, procura-se por disciplinas com aulas a mais. Nesse caso, percorre-se o vetor de períodos, procurando por aulas daquela disciplina. Enquanto a quantidade de aulas daquela disciplina for maior que a necessária, removem-se as aulas dessa disciplina que forem sendo encontradas.

O operador de reparo genético também verifica se existem aulas com professores e/ou turmas iguais sendo ministradas mais de uma vez em um mesmo período. Isso é necessário porque um professor não pode ministrar duas aulas ao mesmo tempo. De forma análoga, uma turma não pode assistir mais de uma aula em um mesmo período. Nesse caso, as aulas que violão essas restrições são colocadas em outros períodos selecionados aleatoriamente, onde essas restrições não sejam violadas.

5.10. Condição de parada

O algoritmo foi projetado para encerrar a sua execução quando for definido que ele já convergiu em um conjunto de soluções, e que novas gerações de indivíduos muito provavelmente não apresentarão melhorias dentro de um intervalo de tempo não proibitivo.

Para isso, a cada geração, verifica-se se o *fitness* da melhor solução da geração atual é igual ao fitness da melhor solução encontrada até o momento. Mantém-se um contador com o número de gerações que se passaram desde que houve modificação no *fitness* da melhor solução. Caso a quantidade de gerações que se passaram sem que houvesse uma melhora alcance um valor maior ou igual a um limite, o algoritmo encerra a sua execução. Caso seja encontrada uma solução cujo *fitness* é maior que o da melhor solução encontrada até o momento, é atribuído ao contador o valor de zero, e o algoritmo continua.

O número limite de gerações que se passaram sem que houvesse uma melhor com relação à melhor solução pode ser definido dinamicamente, como uma proporção da quantidade total de gerações. Dessa forma, quanto mais gerações, mais o algoritmo espera para verificar se uma solução melhor será encontrada.

Entretanto, esse não pode ser o único critério de parada, uma vez que nas primeiras gerações, a quantidade total de gerações que se passou é muito pequena. Isso faz com que o contador atinja o limite especificado com facilidade, o que resulta em um fim prematuro do algoritmo: sua execução é encerrada quando ele ainda poderia encontrar soluções expressivamente melhores caso continuasse executando. Para se resolver esse problema, foi definido um número mínimo de gerações que devem ser criadas. Assim, mesmo que o contador atinja o limite que foi definido, o algoritmo não encerrará sua execução até que se atinja aquela quantidade mínima de gerações.

6. Testes e Resultados

O algoritmo proposto foi implementado utilizando a linguagem C++11, com o auxilio das estruturas de dados presentes na *Standard Template Library*(STL) [Carvalho 2019b]. Para auxiliar a visualização da saída do algoritmo, foi utilizada a biblioteca *JSON for Modern C++*, com objetivo de converter a saída do algoritmo para o formato *JSON* e utilizá-lo

Algorithm 2 Condição de parada

$contador \leftarrow 0$
while $(contador < limite) \vee (contador \leq qtd\_geracoes/proporcao)$ **do**
  if $melhor\_fitness\_total = melhor\_fitness\_atual$ **then**
    $contador \leftarrow contador + 1$
  else
    $contador \leftarrow 0$
  end if
  $criar\_nova\_geracao()$
end while

como entrada para uma aplicação *web* que exibe a grade semanal de horários de forma mais organizada e legível[Carvalho 2019a]. Esta aplicação *web* foi escrita utilizando a linguagem *Javascript* com a biblioteca *React*. Os testes foram realizados em um computador com processador Intel Core i5 8300h e 8 GB de memória RAM DDR4. O sistema operacional utilizado foi o Linux Mint 19.1.

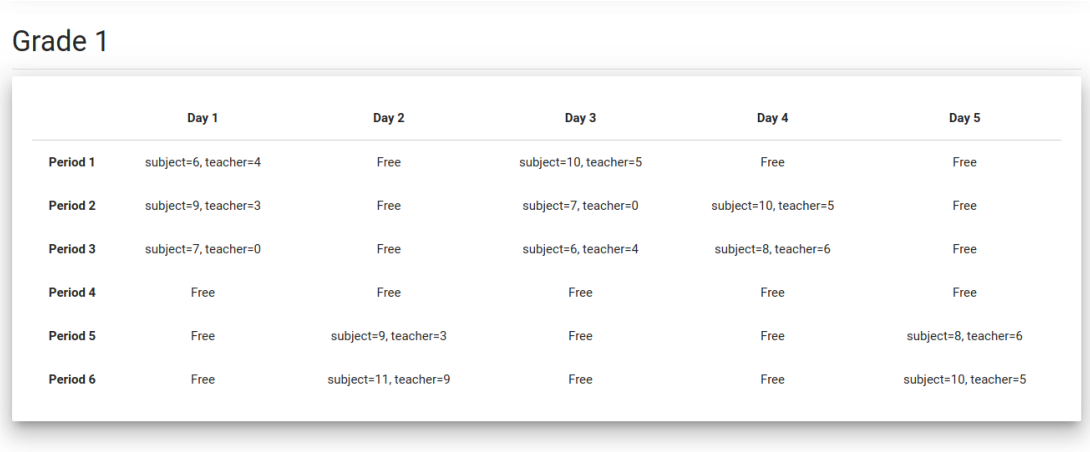

Figura 3. Exemplo de grade horária gerada pelo algoritmo visualizada na aplicação web

Foram utilizadas diferentes populações para os testes, e a proporção utilizada na condição de parada foi de metade do total de gerações. O valor máximo do *fitness* que uma solução pode atingir é de 1000. Em outras palavras, uma solução com *fitness* 1000 é uma solução ótima e não viola nenhuma restrição. 70% da população é gerada através do *crossover*, e os outros 30% por mutações. Dentre os indivíduos gerados através de mutações, é decidido aleatoriamente se o operador de troca de aulas ou troca de professores será utilizado, portanto, cada um tem chances iguais de serem escolhidos. Esses valores foram obtidos experimentalmente e apresentaram os melhores resultados.

Para os testes, foram utilizados dados do curso de Bacharelado em Ciência da Computação do semestre de 2018.2. Nesse semestre haviam 5 turmas, 24 disciplinas e 9 professores. As preferências dos professores foram definidas com base nas disciplinas que eles geralmente assumem no curso e sua área de formação. O algoritmo foi testado com populações de de 25, 35 e 50 indivíduos. Para cada um desses cenários, ele foi executado 20 vezes.

Três propriedades dos resultados estão sendo consideradas para efeito de comparação. Elas são: A quantidade total de gerações que o algoritmo levou até convergir (quanto menor, melhor); *Fitness* da melhor solução encontrada (quanto maior, melhor); Tempo total de execução do algoritmo (quanto menor, melhor). Os valores relacionados levam em consideração todas as 20 execuções para aquela quantidade de indivíduos.

Tabela 3. Resultados para 25 indivíduos

	Gerações	**Fitness**	**Tempo(s)**
Mínimo	1764	837	8.95
Máximo	11102	938	55.86
Média	4284.8	906.8	21.78
Melhor resultado	4154	938	21.28

Tabela 4. Resultados para 35 indivíduos

	Gerações	**Fitness**	**Tempo(s)**
Mínimo	1558	851	11.076
Máximo	7014	944	50.27
Média	3032.25	905.3	21.74
Melhor resultado	3356	944	24.12

Tabela 5. Resultados para 50 indivíduos

	Gerações	**Fitness**	**Tempo(s)**
Mínimo	1520	864	15.47
Máximo	8236	946	82.7
Média	3865.2	912.45	39.207
Melhor resultado	5510	946	55.72

Ao analisar os resultados obtidos, pode-se constatar que que não existem muitas diferenças práticas entre testes realizados com 25 e 35 indivíduos: em média, o tempo de execução e o *fitness* apresentaram diferenças irrisórias. Porém,entre 35 e 50, nota-se um aumento de em média apenas 0,78% do *fitness* e um aumento considerável de 44.55% no tempo total de execução.

7. Considerações finais

Neste trabalho, foi aplicado um algoritmo genético para a resolução do STP para a Universidade Estadual do Piauí.

O algoritmo genético utilizado usa um operador de reparo genético para as soluções inviáveis, além de utilizar dois operadores de mutação no processo de criar novas gerações, juntamente com o *crossover*, além de conservar novas soluções caso elas sejam melhores que as soluções da geração atual.

Todas as soluções encontradas pelo algoritmo são factíveis e próximas de uma solução ótima. Além disso, o algoritmo apresenta um baixo tempo total de execução, o que faz com que ele realmente possa ser aplicado na prática para produzir soluções reais que facilitam o trabalho dos coordenadores de curso.

Para trabalhos futuros, sugere-se dividir este algoritmo em duas fases, como em [Raghavjee and Pillay 2010], e também utilizar processamento paralelo para se obter um melhor desempenho, além de se experimentar outras técnicas de *crossover*, seleção e mutação.

Referências

Abramson, D. and Abela, J. (1991). *A parallel genetic algorithm for solving the school timetabling problem*. Division of Information Technology, CSIRO.

Aragão, A. P. d. (2010). Utilização de algoritmos genéticos para a elaboração de um programa gerador de grade horária.

Burke, E. K. and Newall, J. P. (1999). A multistage evolutionary algorithm for the timetable problem. *IEEE transactions on evolutionary computation*, 3(1):63–74.

Carvalho, J. (2019a). Aplicação web para visualizar a saída do algoritmo. `https://github.com/JeffersonCarvalh0/ga-output-visualizer`. [Acesso em 9 de Julho de 2019].

Carvalho, J. (2019b). Implementação do algoritmo. `https://github.com/JeffersonCarvalh0/tcc`. [Acesso em 9 de Julho de 2019].

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

Nunes, R., Guimarães, N., and Carvalho, C. (2013). Planejamento de grade de horário em uma universidade brasileira usando algoritmos genéticos. In *X Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*.

Poulsen, C. J. B. (2012). Desenvolvimento de um modelo para o school timetabling problem baseado na meta-heurística simulated annealing.

Raghavjee, R. and Pillay, N. (2010). Using genetic algorithms to solve the south african school timetabling problem. In *2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pages 286–292. IEEE.

Rothlauf, F. (2011). *Design of modern heuristics: principles and application*. Springer Science & Business Media.

Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

Sousa, V. N. d., Moretti, A. C., and Podestá, V. A. d. (2008). Programação da grade de horário em escolas de ensino fundamental e médio. *Pesquisa Operacional*, 28(3):399–421.