UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Gabriel Campos Albino

Algoritmos Genéticos Aplicados ao Problema de Alocação de Horários e Professores em Escolas do Ensino Fundamental

Florianópolis
2022

Gabriel Campos Albino

Algoritmos Genéticos Aplicados ao Problema de Alocação de Horários e Professores em Escolas do Ensino Fundamental

Trabalho de Conclusão de Curso do Curso de Graduação em Ciência da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Ciência da Computação.
Orientadora: Profa. Jerusa Marchi, Dra.

Florianópolis
2022

Catalogação na fonte pela Biblioteca Universitária da Universidade Federal de Santa Catarina.

Arquivo compilado às 20:52h do dia 23 de dezembro de 2022.

Gabriel Campos Albino
Algoritmos Genéticos Aplicados ao Problema de Alocação de Horários e Professores em Escolas do Ensino Fundamental : / Gabriel Campos Albino; Orientadora, Jerusa Marchi, Dra.; , - Florianópolis, 20:52, 23 de dezembro de 2022.
60 p.

Trabalho de Conclusão de Curso - Universidade Federal de Santa Catarina, Departamento de Informática e Estatística, Centro Tecnológico, Programa de Graduação em Ciência da Computação.

Inclui referências

1. Inteligência Artificial. 2. Algoritmos Genéticos. 3. Scheduling Problem. 4. Problema do Janelamento. I. Jerusa Marchi, Dra. II. III. Programa de Graduação em Ciência da Computação IV. Algoritmos Genéticos Aplicados ao Problema de Alocação de Horários e Professores em Escolas do Ensino Fundamental

CDU 02:141:005.7

Gabriel Campos Albino

Algoritmos Genéticos Aplicados ao Problema de Alocação de Horários e Professores em Escolas do Ensino Fundamental

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Ciência da Computação" e aprovado em sua forma final pelo Curso de Graduação em Ciência da Computação.

Florianópolis, 23 de dezembro de 2022.

Prof. Jean Everson Martina, Dr.
Coordenador do Curso

Banca Examinadora:

Profa. Jerusa Marchi, Dra.
Orientadora

Prof. Mauro Roisenberg, Dr.
Avaliador
Universidade Federal de Santa Catarina – UFSC

Profa. Luciana Rech, Dra.
Avaliadora
Universidade Federal de Santa Catarina – UFSC

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

AGRADECIMENTOS

Agradeço primeiramente à minha professora e orientadora Jerusa Marchi, pela paciência, cordialidade e companheirismo durante o desenvolvimento deste trabalho.

Agradeço também à minha família, sem a qual não seria possível chegar onde estou.

RESUMO

Este trabalho tem por objetivo auxiliar a comunidade ao propor uma solução ao problema de agendamento de disciplinas e horários, que é recorrente em toda e qualquer instituição de ensino. Foi realizada uma medição de desempenho de um Algoritmo Genético adaptado ao famoso timetabling problem (NP-Completo), visando estabelecer empiricamente os parâmetros ideais de execução e obter um estudo de caso da implementação de variadas técnicas modernas implementadas sobre o algoritmo original.

Palavras-chave: Inteligência Artificial. Algoritmos Genéticos. Problema do Janelamento.

ABSTRACT

This work aims to help the community by proposing a solution to the problem of scheduling subjects and time slots, which is recurrent in any and all educational institutions. A performance measurement of a Genetic Algorithm adapted to the famous timetabling problem (NP-Complete) was carried out, aiming to empirically establish the ideal execution parameters and obtain a case study of the implementation of several modern techniques applied over the original algorithm.

Keywords: Artificial Intelligence. Genetic Algorithms. Timetabling Problem.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Em todo início de um novo período letivo, as instituições de ensino se deparam com o problema de organizar sua grade de horários de forma que não hajam conflitos entre disciplinas de um mesmo período ou choques de disponibilidade na agenda de alunos e professores. Vários requisitos precisam ser atendidos, visto que a grade dos alunos devem sempre estar completas e, ao mesmo tempo, devem satisfazer eventuais preferências ou limitações de agenda por parte dos professores. O fato de tal problema ser reconhecidamente NP-Completo (ULLMAN, 1973) torna a obtenção de uma solução exata e ótima uma tarefa praticamente inviável.

Tendo em vista tais dificuldades, a aplicação de uma heurística provinda da inteligência artificial popularmente conhecida como algoritmos genéticos pareceu adequada para encontrar uma solução aproximada para o problema. Tais algoritmos são amplamente utilizados nas mais diversas áreas do conhecimento e de aplicação, justamente por oferecerem uma solução aproximada e satisfatória para problemas combinatoriais de complexidade elevada (MELANIE, 1999), tais como obter solução de polinômios de alto grau ou o problema do caixeiro viajante.

Ao final deste trabalho, espera-se obter uma análise extensa do desempenho de um Algoritmo Genético adaptado ao problema de tabulação de horários, assim como estimar os parâmetros otimizados para sua execução e explorar técnicas de melhoria de desempenho.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Produzir uma programa que proporcione uma solução aproximada para o problema de janelamento de horários e obter os parâmetros ideais para sua execução. Além disso, serão feitos estudos e medições de desempenho sobre a aplicação de três técnicas utilizadas amplamente em conjunção com algoritmos genéticos para otimização de desempenho: elitismo, heurísticas híbridas e melhorias da população inicial.

1.1.2 Objetivos Específicos

1. Adaptar um algoritmo genético ao problema de janelamento de horários, produzindo uma nova representação de cromossomos;

2. Testar extensivamente sua execução para a análise de comportamento do AG dada a variabilidade dos parâmetros;

3. Avaliar a relevância da aplicação de técnicas de otimização do algoritmo em sua execução.

1.2 ESTRUTURA DO DOCUMENTO

Este documento contará com um capítulo inicial descrevendo o problema de janelamento de horários (do inglês, Timetabling Problem), suas possíveis abordagens e trabalhos correlatos que pro- põem-se a solucioná-lo utilizando AG's. Em seguida, haverá um capítulo destinado a fundamentação teórica de algoritmos genéticos, expondo suas características e variações e descrevendo a abordagem eleita para a realização deste trabalho. Por fim, haverá um capítulo onde serão relatados o processo de execução e os testes realizados com o algoritmo.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 TIMETABLING PROBLEM

Nesta seção será apresentado de forma detalhada o problema central que nos propomos a resolver neste trabalho.

2.1.1 Visão Geral

O problema do encaixe de horários (do inglês, Timetabling Problem) diz respeito a eventos (aulas, exames, etc) que devem ser arranjados em um número de slots, sujeitos a diversas limitações. A necessidade de métodos poderosos e eficientes fica clara ao observar que, para um número x de eventos a serem encaixados em um numero t de slots de tempo, temos x^t possíveis tabulações candidatas a solucionar o problema.

As abordagens mais comuns para a solução de tais problemas consistem simplesmente em achar a tabulação mais curta que satisfaça as restrições impostas utilizando um algoritmo de coloração de grafos (FANG, 1994). Tais algoritmos podem ser executados em tempo polinomial em uma pequena minoria de casos (coloração com duas cores), mas, em sua vasta maioria, são classificados como NP-Completos (GAREY; JOHNSON, 1990).

Existem atualmente ferramentas que permitem criar uma aplicação que resolva o problema de timetabling, mas observa-se certas limitações em seu uso, visto que abordam o problema como uma simples satisfação de condições pré-estabelecidas. Utilizando a linguagem Prolog, por exemplo, pode-se facilmente criar um programa que satisfaça um certo número de restrições e crie uma tabulação adequada. O desempenho da aplicação, porém, não irá atender à necessidade de uma instituição de ensino de porte mediano, visto que o tempo de execução irá crescer de forma exponencial para cada entidade a ser alocada.

2.1.2 Exemplo

A seguir, veremos um exemplo hipotético do timetabling problem. Assuma que existam dez estudantes (s1, s2, ..., s10), seis exames (e1, e2, ..., e6), e oito slots de tempo (t1, t2, ..., t8) tais que t1 até t4 ocorram no primeiro dia e t5 até t8 ocorram no segundo dia. Cada dia possui quatro slots de tempo, com dois de manhã e dois pela tarde. As tabelas a seguir ilustram o problema:

As retrições neste caso são que um estudante não pode realizar dois exames diferentes em um mesmo slot de tempo e realizar três exames em um mesmo dia deve ser fortemente evitado. Estudantes realizando dois testes consecutivos também não é algo desejável. Uma possível solução para o dado problema é mostrado na tabela a seguir, onde as condições de aceitação são observadas:

Estudante	Exame
s1	e1,e2,e4,e6
s2	e1,e2,e4,e5
s3	e1,e2,e4,e5
s4	e1,e2,e5
s5	e4
s6	e4,e5
s7	e2,e4
s8	e2,e4,e6
s9	e3,e4,e6
s10	e3,e4,e6

Tabela 1 – Tabela de estudantes e respectivos exames.

Dias	Manhã		Tarde	
	09:30-11:00	11:30-13:00	14:00-15:30	16:00-17:30
Dia 1	t1	t2	t3	t4
Dia 2	t5	t6	t7	t8

Tabela 2 – Tabela de slots de tempo.

Dias	Manhã		Tarde	
	09:30-11:00	11:30-13:00	14:00-15:30	16:00-17:30
Dia 1	e1	e3		e4
Dia 2	e2		e5	e6

Tabela 3 – Uma possível solução do problema.

Para o exemplo apresentado, uma solução é facilmente obtida até mesmo sem o auxílio de um computador. Para um número de estudantes que se aproximem de uma centena, porém, pode-se imaginar que os casos a serem analisados crescem de forma exponencial, fatalmente requerendo um poder computacional elevado para se obter uma resposta.

O exemplo acima foi apresentado com o intuito de aprofundar o entendimento acerca do problema de tabulação de horários e suas características em termos gerais. Para o caso específico deste trabalho, a instância será relativamente mais simples, visto que a solução consiste apenas em criar uma grade-horária para professores ministrarem suas aulas em uma escola de ensino fundamental com poucas turmas.

2.2 ALGORITMOS GENÉTICOS

Esta seção visa esclarecer de forma genérica os conceitos fundamentais envolvidos nos algortimos genéticos e suas diversas aplicações.

2.2.1 A Analogia

Similarmente às redes conexionistas e outros procedimentos provindos da área da inteligência artificial, algoritmos genéticos também se originaram de uma analogia com processos biológicos naturais. Tais "algoritmos" tentam resolver problemas genéricos seguindo os princípios básicos da teoria evolutiva, trabalhando com conceitos fundamentais como genes, cromossomos, indivíduos, população, etc. A ideia central é selecionar diretamente as soluções mais adequadas para determinado problema (similarmente ao processo da evolução das espécies) e melhorá-las a cada iteração, eliminando assim a necessidade de se obter um método único para a solução de um problema específico (MELANIE, 1999).

Em sua obra A Origem das Espécies, Charles Darwin faz uma breve reflexão do poder do processo evolutivo. Embora originada em outro contexto, tal reflexão se encaixa quase que perfeitamente no âmbito da computação evolucionária, se levados em conta a infinidade de problemas que podem ser solucionados com esta simples, porém eficiente técnica.

> "Que limites podemos impor a este poder, que age por longas eras e escrutina toda constituição, toda a estrutura e todos os hábitos de cada criatura - favorecendo o que é bom e rejeitando o que é ruim? Não vejo limites a esse poder de adaptar vagarosa e maravilhosamente cada forma às mais complexas relações de vida".
>
> - Charles Darwin

Em resumo, Algoritmos Genéticos imitam, ainda que ingenuamente, o processo que moldou a evolução dos seres vivos ao longo das eras, e que, de forma impessoal e pragmática, elimina os mais fracos e prevalece os mais fortes, como uma força incontrolável que impulsiona incessantemente as formas de vida rumo a um objetivo desconhecido.

2.2.2 O Algoritmo

Antes de tudo, vale ressaltar que o nome de "Algoritmos Genéticos"é utilizado popularmente e não se encaixa no rigor da definição formal de um algoritmo. Tais procedimentos não possuem uma parada garantida e, por proverem apenas uma solução aproximada, dependem de uma condição de parada arbitrariamente definida para alcançar o fim de sua execução (E. HOPCROFT; D. ULLMAN, 1969). Adotaremos para os fins deste trabalho a nomenclatura popular e mais comumente usada e, como referencial teórico para os conceitos abordados, adotaremos o livro "An introduction to Genetic Algorithms"de Mitchell Melanie (MELANIE, 1999).

Algoritmos Genéticos podem geralmente ser subdivididos em duas etapas principais: geração da população inicial e evolução. Em sua primeira etapa, as potenciais soluções iniciais do domínio do problema são codificadas em representações que devem suportar as variações e operações de seleção necessárias; geralmente, essas representações são tão

simples quanto uma cadeia de bits (como mostrado na figura 1), apesar de haverem representações mais complexas para problemas específicos.

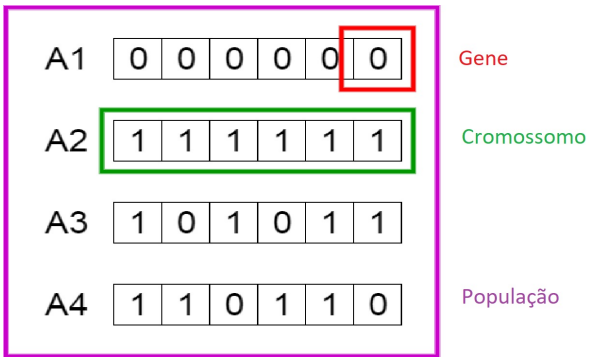

Figura 1 – Representação das entidades em um algoritmo genético

Assim como na biologia natural, as entidades presentes em um algoritmo genético seguem uma hierarquia: um gene diz respeito à uma característica específica - um genótipo - e o conjunto de genes forma um cromossomo (ou, no caso da analogia, um indíviduo). Cada indivíduo é formado por características únicas que serão avaliadas posteriormente, e o conjunto de indivíduos é denominado população. O objetivo final do algoritmo é prover uma população que contenha ao menos um indivíduo que seja apto o suficiente de acordo com algum parâmetro pré-estabelecido, avaliado através de seus genes.

Na segunda etapa, os algoritmos de acasalamento e mutação, análogos à atividade reprodutiva de formas vivas biológicas, produzem uma nova geração de indivíduos que recombinam o material genético de seus pais. Para isso, primeiro faz-se uma análise dos indivíduos (de acordo com uma função de aptidão adequada ao domínio do problema) que posteriormente poderão ser escolhidos para o acasalamento, com probabilidade de escolha proporcional à sua aptidão. Tal distribuição probabilística de indivíduos selecionados para gerações posteriores é um fator importante, já que garante uma chance pequena de até mesmo os mais “fracos” da população poderem se reproduzir (se eles fossem simplesmente apagados, informações ou combinações cruciais poderiam ser perdidas).

Abaixo é mostrada uma versão em pseudo-linguagem do "esqueleto"de um algoritmo genético, inspirada na versão do livro de Mitchell. Nele, é possível observar na linha 4 a função Initialize P(t), responsável por criar a representação estrutural e lógica da população (primeira etapa), enquanto que na linha 6 a função Evaluate P(t) se encarrega de avaliar cada indivíduo da população e atribuir-lhes um grau de aptidão, dando início à segunda etapa do algoritmo.

```
Begin
t <- 0;
//Cria a populacao inicial
Initialize P(t);
//Atribui valores a populacao segundo a funcao de aptidao
```

```
Evaluate P(t);

while (condicao de parada nao alcancada) do
Begin
    //Incremento de uma geracao
    t <- t+1;
    //Etapa de selecao
    Select P(t) from P(t-1);
    //Etapa de acasalamento
    Crossover P(t)
    //Aplicacao da mutacao
    Alter P(t);
    //Atribui os valores de apticao a nova populacao
    Evaluate P(t);
End;
End.
```

A primeira fase da segunda etapa é denominada seleção e consiste em aplicar um método probabilístico para escolher os indíviduos que irão compor a população da próxima geração. Existem diversos métodos probabilísticos de escolha de indivíduos, sendo os métodos da roleta e do torneio os mais amplamente difundidos e utilizados (LIPOWSKI; LIPOWSKA, 2011).

O método da roleta apresenta uma chance de determinado indivíduo ser selecionado dada pela equação (onde f é a função de aptidão, v o indivíduo e n o tamanho da população):

$$p_i = f(v_i) / \sum_{j=1}^{n} f(v_j) \tag{1}$$

Pela equação 1, vemos que a chance de um indivíduo ser sorteado é diretamente proporcial à sua aptidão. Isso faz com que gerações futuras quase certamente tenham melhores soluções que a geração atual e ao mesmo tempo garante uma chance dos piores indivíduos também serem selecionados.

O método do torneio consiste em selecionar indivíduos aleatoriamente na população, dois a dois. Então, para cada par sorteia-se um número aleatório R, seguindo uma distribuição uniforme. Se $R < K$ (onde K é um parâmetro, como 0.75, por exemplo), o indivíduo mais apto é selecionado, caso contrário a seleção prevalece o menos apto. Quanto maior é o valor de K, maior será a pressão seletiva imposta à população (SHUKLA; PANDEY; MEHROTRA, 2015).

Cada domínio de problema, juntamente com sua respectiva representação, irá apresentar um método ótimo diferente para a escolha de futuras gerações. Uma característica

que todos os métodos devem ter em comum, porém, é a garantia de uma chance, ainda que pequena, de indivíduos com aptidão “ruim” serem selecionados para compor a próxima geração. Tal característica é importantíssima para garantir que traços genéticos potencialmente essenciais para a composição da solução ideal não sejam perdidos, além de evitar uma possível convergência precoce da população.

A fase de “acasalamento” consiste na escolha de um operador genético para realizar a troca de “genes”. O operador mais comum é a recombinação (ou crossover), que basicamente toma duas soluções candidatas e as divide, seja exatamente na metade ou em algum ponto arbitrário, originando dois filhos, cada um contendo sua primeira metade de genes vinda de um genitor e outra metade vinda do outro. A figura 2 ilustra uma possível aplicação deste operador genético (a imagem é meramente ilustrativa e não reflete a representação que será utilizada neste trabalho).

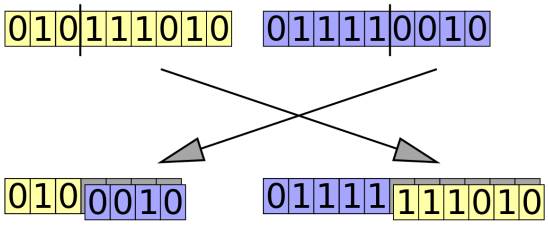

Figura 2 – Exemplo de uma aplicação hipotética do crossover

Durante a segunda etapa, existe ainda a possibilidade de ocorrer uma "mutação", que é um operador genético que toma um único candidato e troca de forma aleatória algum de seus aspectos. Este operador é de suma importância, visto que uma população inicial pode não conter um componente essencial da solução.

A segunda etapa do algoritmo irá então se repetir até que algum critério de parada pré-estabelecido seja alcançado (como, por exemplo, se a função de aptidão alcançar certo limiar).

2.2.3 Variações

Apesar de oferecerem um método prático e fácil para solucionar problemas de busca e otimização de forma relativamente eficiente, os algoritmos genéticos também possuem suas limitações. Se o número de elementos expostos à mutação for alto, por exemplo, a variabilidade da população aumentará demais, causando um efeito de perda de informações advindas da seleção e cruzamento, efeito este exatamente oposto ao que se pretende realizar. Outro ponto negativo é que soluções “boas” serão relativas somente a outras soluções obtidas, sem nunca haver uma corretude absoluta ou resposta definitiva para dado problema.

Para mitigar o problema do custo computacional, muitas tentativas de aprimorar a heurística evolutiva foram realizadas. Em uma delas (DEB, 2002), sugeriu-se trabalhar

com uma versão elitista dos algoritmos genéticos, chamada de NSGA-II, onde o melhor cromossomo da geração atual estaria também presente na próxima. Este novo algoritmo reduziu consideravelmente a complexidade de seu predecessor pela dominação de indivíduos melhores e por utilizar uma regra de compartilhamento para empates que dá preferência à diversidade. Isto permitiu alcançar uma convergência de resultados de forma mais rápida e prevenir o esquecimento catastrófico de soluções boas previamente encontradas. O elitismo será explorado mais a fundo no capítulo 5.

Existem ainda os chamados algoritmos genéticos baseados em gênero (BULLINARIA, JOHN A., 2003), que dão um passo além na analogia biológica, e dividem a população em duas categorias, aplicando diferentes formas de seleção sobre cada uma. Neste método, existe apenas um subgrupo que efetivamente compete entre si para conseguir a "aprovação" de algum indivíduo do outro grupo e acasalar, semelhante ao comportamento observado entre machos e fêmeas de diversas espécies na natureza. A parte não-competitiva da população permite diversificar o espaço de busca e previne a convergência prematura. AG's baseados em gêneros apresentaram um desempenho superior aos AG's comuns para algumas classes de problemas, tais como o problema de configuração automática de algoritmos (TIERNEY, 2009).

2.2.4 Paralelização

Algoritmos genéticos possuem uma vantagem inerente no que diz respeito ao seu grande potencial de paralelização. Na etapa do cálculo de aptidão, visto que cada indivíduo é tratado como uma entidade independente das outras, é possível fazer a atribuição dos valores de forma completamente concorrente sobre a população. Já na etapa de crossover, os "casais"que irão trocar genes também podem ser tratados de forma isolada sem afetar o resultado final.

Desta forma, surgiu assim a ideia de dividir o que seria um só algoritmo rodando sobre uma população inteira, e criar uma variante na qual existiriam várias versões de um mesmo algoritmo, porém diferindo entre si em vários aspectos, cada um executando sobre um subconjunto da população total (D. BETHKE, 1976). A cada repetição do algoritmo, os subgrupos podem ter seus indivíduos trocados entre si (migração), aumentando de forma considerável a diversidade da amostra e contribuindo para uma convergência mais rápida e precisa. A única característica idêntica que os algoritmos paralelos devem ter é a função de aptidão, para assim comparar os valores e se aproximar da solução ideal. Desta forma, os algoritmos executando paralelamente podem diferir nos seguintes aspectos, entre outros:

1. Como indivíduos são criados
2. Como ocorre a mutação e como indivíduos são escolhidos para mutar
3. Como ocorre o crossover

4. Quantos indivíduos sobrevivem a cada iteração

5. Como indivíduos são selecionados

Com a demanda por poder computacional aumentando nos últimos anos, foram criados também algoritmos genéticos distribuídos, projetados para executar sobre quantidades massivas de dados, ou em situações onde os indivíduos são codificados em representações muito complexas (ALBA; M. TROYA, s.d.). A ideia é a mesma dos algoritmos paralelos, adicionando, porém, uma camada a mais de concorrência. Tal aplicação é capaz de tomar proveito dos recursos escaláveis e robustos que a computação em nuvem proporciona.

Para os fins deste trabalho, o projeto de um algoritmo genético paralelo que execute sobre uma quantidade relativamente pequena de processadores (no máximo quatro) será suficiente para obter uma boa solução com um desempenho aceitável.

3 ESTADO DA ARTE

Este capítulo destina-se a avaliar o estado da arte acerca de algoritmos genéticos aplicados ao timetabling problem, listando trabalhos anteriores e evidenciando sua correlação com este trabalho.

3.1 TRABALHOS RELACIONADOS

Utilizar algoritmos genéticos para a solução do problema de tabulação de horários não é algo novo. As primeiras tentativas datam de 1990, quando um grupo de pesquisadores (CORLONE; DORIGO; MANIEZZO, 1990) construiu uma tabulação para as aulas de uma escola secundária italiana usando a abordagem de AG's e reportaram a primeira aplicação desse tipo a ser bem-sucedida. A representação escolhida era formada por uma quíntupla, consistida por um recurso (professor), intervalos de tempo (horas), aulas, uma matriz (a tabulação em si) e uma função a ser maximizada. Cada linha da matriz representava um professor, cada coluna era um horário e em cada célula era alocada uma aula. O resultado foi uma aplicação que satisfazia todas as limitações obrigatórias, enquanto minimizava a violação das limitações secundárias e opcionais.

Em 1994, surgiu a ideia inovadora de usar um algoritmo de coloração de grafos para primeiramente criar uma tabulação factível que satisfizesse todas as condições obrigatórias (que eram: um estudante não pode realizar dois exames ao mesmo tempo e as salas de aula devem acomodar todas os estudantes que realizariam a prova), para somente então operar um algoritmo genético sobre a tabulação resultante (BURKE; ELLIMAN; WEARE, 1994). Este trabalho deu origem à uma aplicação de desempenho satisfatório, mas acarretou em uma complexidade muito grande na escolha de indivíduos para o crossover e um tratamento a mais precisou ser feito nos casos em que a tabulação se tornava não-factível.

Também em 1994, investigou-se o uso de representações indiretas para o timetabling problem. Nesta abordagem, os cromossomos codificam instruções para a construção de uma tabulação, ao invés de representar a entidade em si. A ideia geral é que os cromossomos armazenem informações sobre qual evento agendar em seguida, e onde alocá-lo. Caso tal evento não possa ser alocado sem violar uma condição obrigatória, o cromossomo codificaria onde procurar por um espaço alocável e factível (PAECHTER *et al.*, 1994). A aplicação resultante tende a trabalhar com um espaço de busca muito menos esparso que o normal, mas, consequentemente, isso leva à possibilidade de uma provável resposta ótima nunca ser encontrada.

Um AG foi usado para resolver pequenas e grandes instâncias do problema de timetabling por Zhong (ZHONG *et al.*, 2013). O estudo modificou alguns operadores genéticos básicos que restringem a criação de novos conflitos em cada indivíduo para melhorar a performance. O estudo reduz problemas de agendamento com variáveis binárias de tamanho massivo para um tamanho aceitável, eliminando certas dimensões dos

problemas e transformando-as em restrições. A redução do tamanho de cada gene é feita agrupando vários bits em um valor de gene. Tal técnica tem a possibilidade de resolver o problema de tamanho completo e melhora a velocidade do algoritmo em dezenas de segundos. Utilizando operadores inteligentes, o algoritmo converge muito mais rápido do que o algoritmo básico e, portanto, o estudo representa um bom ponto para resolver o timetabling problem.

Mais recentemente, um Algoritmo Genético Dinâmico (DGA) foi proposto por Sastry (SASTRY; D.E.; KENDALL, 2014) que simultaneamente usa mais de um operador de crossover e mutação para dar origem à novas gerações. As taxas de mutação e de crossover são adaptáveis e mudam de acordo com a avaliação da função de aptidão de cada indivíduo. Esta nova técnica apresentou uma performance superior à de abordagens anteriores e se mostrou escalável com uma quantidade de processadores maior.

Em 2015, um estudo de caso sobre o timetabling problem utilizando AG's foi realizado por Gonsalves e Oishi (GONSALVES; OISHI, 2015). O estudo conclui que algoritmos genéticos podem ser aplicados em domínios onde não há conhecimento suficiente ou o tamanho e a complexidade das entradas são muito altos. O crossover é realizado com uma quantidade de bits predefinida, geralmente 1. No entanto, o estudo descobriu que esta forma de troca de genes pode levar à convergência da prole em vez de calcular uma solução onde os indivíduos chegariam a um nível de fitness adequado. Neste caso, utiliza-se a mutação para evitar tal convergência prematura. O estudo sugere ainda outras melhorias dos parâmetros, como a quantidade de crossovers e a chance individual de mutação.

Mais recentemente, um grupo de pesquisadores propôs uma abordagem híbrida única para a solução do timetabling problem em universidades (REZAEIPANAH; MATOOR; AHMADI, 2020). A versão foi denominada IPGALS (do inglês, Improved Parallel Genetic Algorithm and Local Search) e combina paralelismo, algoritmos genéticos e métodos de busca local para minimizar o tempo de execução do programa. A inovação desta proposta reside na forma da qual a busca local foi implementada, atuando logo após a fase de acasalamento com o intuito de melhorar o genes, reintroduzindo possíveis traços genéticos perdidos ou pouco usados e resolvendo conflitos em meio à execução padrão do algoritmo genético. Os resultados reportados foram satisfatórios para instâncias pequenas do problema, mas o IPGALS não foi capaz de gerar soluções factíveis para instâncias maiores.

A seguir, apresentamos a tabela 4 que sumariza os trabalhos estudados e evidencia suas principais contribuições/inovações.

3.2 CORRELAÇÕES

Neste trabalho, nos basearemos na representação utilizada pelo grupo de pesquisadores italianos (CORLONE; DORIGO; MANIEZZO, 1990). Da quíntupla utilizada pelo grupo, utilizaremos três aspectos em comum: o número de slots (m), o número de profes-

Autor	Ano	Contribuição
Corlone	1990	Primeira aplicação bem-sucedida
Burke	1994	Melhoria da população inicial
Paechter	1994	Representações indiretas (instruções)
Zhong	2013	Redução do problema pelas restrições
Sastry	2014	Taxas dinâmicas de crossover e mutação
Gonsalves	2015	Estudo sobre o estado da arte
Ahmadi	2020	IPGALS (algoritmo híbrido paralelo)

Tabela 4 – Tabela de trabalhos relacionados.

sores (n) e a matriz de tabulação (mxn), que é a grade de horários em si. A representação eleita será apresentada em detalhes em uma seção posterior deste documento.

Em sintonia com o trabalho de Burke (BURKE; ELLIMAN; WEARE, 1994), será avaliada uma versão semelhante de melhoria do estado inicial do algoritmo. Porém, ao invés de criar uma tabulação factível - que, no caso deste trabalho em específico, levaria um tempo de execução extremamente elevado - iremos apenas otimizar a população inicial a fim de se obter a menor quantidade de conflitos possível para então ser executado o AG normalmente.

Será testada também uma versão híbrida do algoritmo genetico, relativamente semelhante ao IPGALS (REZAEIPANAH; MATOOR; AHMADI, 2020). Aplicaremos um algoritmo de busca local ao final da execução do AG, no momento que forem obtidas instâncias de grade-horárias onde existam apenas dois conflitos ou menos. Apesar do IPGALS ser uma técnica mais robusta e com aplicação em etapas diferentes da nossa, esperamos obter uma medição equivalente para esta versão de algoritmo híbrido.

4 A ABORDAGEM ELEITA

Neste capítulo serão descritas em detalhes cada etapa do algoritmo, assim como especifidades sobre a representação, restrições e cálculo de aptidão.

4.1 RESTRIÇÕES

Em relação as restrições, deverão ser obrigatoriamente notadas (restrições rígidas):

1. O mesmo aluno não pode assistir a duas aulas ao mesmo tempo
2. O mesmo professor não pode ministrar duas aulas ao mesmo tempo
3. O número de aulas deve menor ou igual ao número de slots disponíveis

Existem ainda aspectos que são desejáveis, mas não obrigatórios (restrições brandas), tais como:

1. Duas aulas da mesma disciplina não devem acontecer imediatamente uma após a outra para uma mesma turma
2. Aulas de educação física devem ser evitadas em horários logo em seguida às refeições

4.2 REPRESENTAÇÃO

Para o uso correto de um AG aplicado a qualquer problema, o primeiro passo importante é eleger uma representação adequada da entidade em forma de cromossomo. Neste trabalho, utilizaremos uma representação que consiste em codificar cada professor da instituição de ensino em uma longa cadeia de números inteiros, onde cada posição da mesma representaria um horário de aula e cada número inteiro corresponderia a uma turma específica. Como em escolas do ensino fundamental o mais comum é haverem quatro aulas por turno, o horário semanal de um determinado professor que só ensina no período da manhã seria traduzido em uma string com 20 posições (5 dias da semana).

Tal representação foi inspirada no trabalho de pesquisadores italianos citado no capítulo 3 (CORLONE; DORIGO; MANIEZZO, 1990), visando manter a simplicidade (a título de performance) e ao mesmo tempo atender todas as necessidades que uma grade-horária de ensino fundamental requer.

Na figura 3, onde observamos as cadeias de 20 inteiros referentes a dois possíveis professores, podemos notar que o número quatro aparece na quarta posição em ambos os casos. Isso significa que a turma 4 foi atribuída a dois professores distintos ao mesmo tempo (no quarto horário da semana, ou seja, no quarto horário da segunda-feira). Isso violaria uma restrição rígida do nosso algoritmo e esta solução receberia uma penalidade elevada na etapa de cálculo de aptidão.

Ana = [285401564631123467 51]

Jorge = [378436458713214376 15]

Figura 3 – Representação dos horários de dois professores. Os 0's representam janelas de tempo livre na grade.

	Segunda	Terça	Quarta	Quinta	Sexta
1º	Turma 2		Turma 4	Turma 1	Turma 6
2º	Turma 8	Turma 1	Turma 6	Turma 2	Turma 7
3º	Turma 5	Turma 5	Turma 3	Turma 3	Turma 5
4º	Turma 4	Turma 6	Turma 1	Turma 4	Turma 1

Tabela 5 – Grade de horários traduzida para a professora Ana.

Na tabela 5, temos a tradução da cadeia de inteiros da professora Ana em uma grade de horários pronta para o uso diário. Como cada turma necessariamente deve ser atendida com o mesmo número de aulas independente do ano em que os alunos se encontram, não se faz necessário um tratamento adicional para escolas onde existam mais de uma turma por ano. Uma simples tradução interna pode ser feita ao final da execução para atribuir a identificação adequada à determinada turma.

Nesta forma de representação, um indivíduo consiste no conjunto das grades de todos os professores, configurando assim uma possível solução para o problema (figura 4). A população, por sua vez, é representada por um conjunto de cem indivíduos. O programa foi estruturado com uma classe simples para englobar os atributos de um professor (tais como nome, disciplina lecionada, etc) e outra classe composta por uma lista de professores e sua aptidão, como mostra o diagrama UML exposto na figura 5.

```
[0, 0, 6, 1, 0, 3, 0, 6, 5, 0, 2, 0, 1, 0, 4, 5, 4, 2, 3, 0]
[6, 0, 0, 0, 1, 1, 2, 5, 4, 3, 3, 0, 2, 0, 0, 6, 5, 0, 4, 0]
[3, 0, 3, 0, 0, 2, 0, 1, 3, 2, 0, 2, 3, 0, 1, 2, 0, 1, 1, 0]
[4, 6, 0, 5, 0, 6, 4, 0, 0, 0, 4, 0, 4, 5, 5, 0, 0, 6, 6, 5]
[5, 5, 0, 3, 3, 4, 0, 0, 1, 4, 1, 6, 6, 2, 0, 0, 0, 0, 0, 2]
[2, 1, 1, 0, 0, 0, 3, 0, 2, 0, 0, 0, 0, 1, 2, 1, 3, 3, 2, 3]
[0, 4, 5, 0, 0, 0, 6, 0, 0, 0, 6, 5, 0, 4, 6, 4, 6, 5, 5, 4]
[1, 2, 2, 0, 5, 0, 0, 0, 6, 6, 0, 4, 5, 3, 3, 0, 1, 4, 0, 0]
[0, 3, 0, 6, 4, 0, 0, 2, 0, 5, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 4, 2, 0, 1, 3, 0, 0, 5, 0, 0, 6, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 5, 0, 4, 0, 1, 0, 3, 0, 0, 0, 0, 2, 0, 0, 6]
[0, 0, 4, 2, 6, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 1]
```

Figura 4 – Matriz representando a grade semanal de todos os professores contidos em uma solução (indivíduo).

Cabe ressaltar que a solução final apresentada pelo algoritmo consiste em uma grade de horários para cada professor da escola. Para formular a grade horária de uma

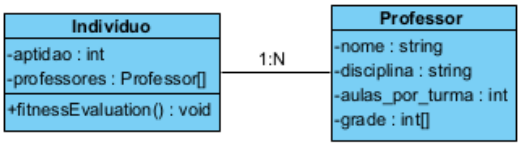

Figura 5 – Diagrama de classes.

turma específica, bastaria percorrer o indivíduo que representa a solução e identificar qual professor dará aula para aquela turma em questão em cada horário da semana.

4.3 GERAÇÃO DA POPULAÇÃO INICIAL

A população inicial de indivíduos neste trabalho foi gerada (antes das adaptações discutidas no capítulo 5) de forma completamente aleatória. Para cada posição da matriz de horários, gera-se um número inteiro aleatório respeitando o intervalo $[0, n]$, onde n é o número de turmas da escola. Daí, verifica-se com o auxílio de um contador se o professor já atendeu o número de aulas por semana para aquela turma sorteada. Se sim, repete-se o processo até sortear uma turma que ainda não foi atendida. Se não, o horário é alocada para aquela determinada turma com sucesso.

Caso o número zero seja sorteado, antes de alocá-lo verifica-se se o professor já atingiu seu número máximo de janelas por semana, que é calculado por $Slots - Turmas \times Aulas/Turma$.

Existem na literatura abordagens que codificam restrições rígidas para serem atendidas já na fase de geração da população inicial (POTVIN, 1996). Tal método não seria condizente com o trabalho proposto neste documento, visto que gerar uma tabulação de horários sem conflitos já é o problema em si do qual nos propomos a resolver. Gerar uma população que atendesse à esse requisito exigiria um estudo à parte sobre métodos de busca otimizados (como coloração de grafos, por exemplo) e fugiria do escopo previsto. Sendo assim, o objetivo final do algortimo genético neste caso não é otimizar uma vasta gama de soluções, mas sim gerar uma solução factível em primeiro lugar.

4.4 FUNÇÃO DE APTIDÃO

O cálculo da função de aptidão consiste em uma varredura da matriz de horários de um indivíduo, coluna por coluna. Primeiramente, atribui-se a um indivíduo a pontuação máxima (calculada por $n \times 1000$, onde n é o número de turmas) para que então o score seja reduzido a cada conflito identificado ou restrição não observada. A pontuação máxima deve ser alterada em relação ao número de turmas para evitar possíveis divisões por zero no decorrer da execução do algoritmo.

A penalidade para quebra de restrições rígidas é de 50 para cada repetição da mesma turma em um slot de tempo. Se uma turma for alocada apenas uma vez em um slot da semana, o indivíduo não é penalizado. Já se tal turma for alocada duas vezes, a penalidade será de $2 \times 50 = 100$. Caso ela seja alocada três vezes a penalidade será de $3 \times 50 = 150$, e assim por diante.

A função conta com o auxílio de um vetor contador que possibilita alcançar uma execução com complexidade $O(N \times M)$, sem precisar percorrer a mesma coluna da matriz diversas vezes. O algoritmo é apresentado na seção de anexos, ao final deste documento.

4.5 SELEÇÃO

A seleção dos indívíduos aptos para a próxima geração é um fator muito importante para a eficiência e garantia de convergência do algoritmo. Para este trabalho, o método de seleção escolhido foi o da roleta, implementado da mesma maneira que foi explicado no capítulo 2. Este método se provou eficiente e nos permitiu obter resultados satisfatórios para a abordagem eleita.

O método do torneio também foi implementado e seus resultados foram testados de forma preliminar. Tal método, porém, pouco influenciou no tempo de execução total do algoritmo e, por isso, voltamos ao método da roleta original que havia sido eleito para esta etapa, por questão de maior simplicidade.

Foi implementado um elitismo de dez indivíduos, como será descrito em mais detalhes no capítulo 5. Isso significa dizer que, no começo da fase de seleção, identifica-se o melhor indivíduo da geração anterior e automaticamente adiciona-se dez cópias do mesmo à geração atual. Tal técnica se prova importante para obter-se uma convergência mais acentuada para o algoritmo.

4.6 CROSSOVER

Neste trabalho, o crossover é realizado de tal maneira que metade dos genes de um indivíduo são trocados com os de outro. Ou seja, metade dos professores que compõem uma possível solução são trocados. Essa forma de cruzamento nos proporcionou uma boa variabilidade genética e permitiu que a convergência ocorresse de forma gradual, porém rápida. Nesta fase, dos dez indivíduos que vêm da geração anterior por meio do elitismo, apenas nove ficam aptos a trocarem seus genes (para garantir que a melhor solução não será perdida no decorrer da execução do algoritmo). A imagem 6 ilustra quais genes são trocados.

Cabe ressaltar que o crossover, que é realizado após a etapa de seleção, age sobre uma população completa de indivíduos. Primeiramente, sorteia-se um número aleatório para cada indivíduo seguindo uma distribuição uniforme de 0 a 1 e, caso este número seja maior que 0.5 (parâmetro obtido experimentalmente como descrito no capítulo 5), o

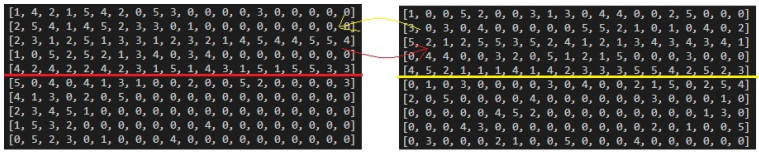

Figura 6 – Ilustração de um cruzamento entre dois indivíduos.

indivíduo é selecionado para o "acasalamento". Então, sorteia-se um número inteiro no intervalo de 2 a 100 (que é o tamanho da população, excluindo-se o primeiro indivíduo para manter-se o elitismo) para representar o indivíduo escolhido como par na troca de genes. Daí então é feita a troca, subtituindo a primeira metade da grade horária de um pelo outro, como mostra a figura 6.

Como a lista de professores de cada indivíduo segue sempre a mesma ordem (se existe uma professora de português chamada Ana, por exemplo, ela sempre estará na mesma posição do vetor em todos os indivíduos da população - propriedade essa advinda da forma da qual a população é iniciada), a troca garante que os genes substituídos no crossover sempre obedeçam às restrições de números de aulas por semana que cada professor deve lecionar.

Foram feitos testes preliminares de crossover onde, ao invés de sortear um número aleatório para cada indivíduo e trocar metade dos seus genes imediatamente, sorteava-se um número para cada linha da tabela de cada indivíduo (ou seja, cada professor que compunha uma solução tinha a chance individual de ser trocado com com o mesmo professor de outro indivíduo). Isso acarretou, porém, em um tempo maior de execução para esta etapa do algoritmo e tal mudança foi descartada.

4.7 MUTAÇÃO

A mutação em nosso algoritmmo se comporta de forma diferente da padrão para algoritmos genéticos. Não seria possível trocar aleatoriamente o valor de um gene, pois isso iria desfazer a configuração necessária do número de aulas que cada professor precisa lecionar e certamente algumas turmas não seriam atendidas. Ao invés disso, os genes são simplesmente trocados de posição na mesma linha da matriz de grade de horários. A figura 7 mostra a grade de um mesmo professor hipotético (linha da matriz) antes e depois de sofrer uma mutação. Assim como na etapa de crossover, apenas nove dos dez indivíduos provindos do elitismo têm a possibilidade de sofrer mutação, para preservar a melhor solução.

Para a realização da mutação, primeiramente sorteia-se um número aleatório para cada indivíduo da população seguindo uma distribuição uniforme entre 0 e 1. Caso o número seja maior que 0.25, o indivíduo é selecionado para mutar. Então, três números inteiros são sorteados para cada indivíduo selecionado: um inteiro no intervalo de $[0, n-1]$ (onde n é o número de professores da escola) para selecionar qual professor será mutado,

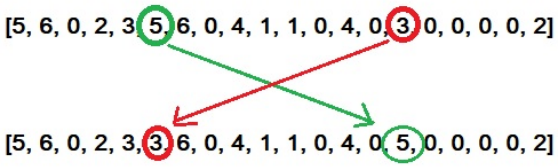

Figura 7 – Ilustração de uma possível mutação ocorrendo sobre a grade de um professor.

e dois inteiros no intervalo de $[0, 19]$ (número de slots na semana), para eleger os slots que serão trocados de lugar.

Desta forma, mesmo com a taxa aparentemente bastante elevada (75%), apenas pouquíssimos genes trocam de fato seus valores na fase de mutação. Essa adaptação se fez necessária para cumprir às restrições de ordem e número de aulas que cada professor deve lecionar. Fazendo-se uma breve comparação com um processo de mutação convencional, onde a taxa é aplicada para cada gene que compõe a solução (e não ao indivíduo como um todo, como é o caso deste trabalho), teríamos uma taxa equivalente descrita pela equação 2 (onde n é o número de professores na escola):

$$T_m = 0,75 \cdot \frac{2}{20 \cdot n} \tag{2}$$

A equação representa a taxa de 75% aplicada sobre dois do total de genes contidos em um individuo. Então, para uma escola com 12 professores (necessário para atender de 6 a 10 turmas), a taxa de mutação equivalente, se considerássemos a chance individual de cada gene, seria de 0,625%.

Tal adaptação na implementação desta etapa do algoritmo nos permitiu obter um tempo de execução menor ao poupar o esforço de calcular um número aleatório e fazer uma comparação para cada gene, ao mesmo tempo que garante boa variabilidade genética e taxa de convergência.

4.8 CRITÉRIO DE PARADA

O algoritmo é executado, para os fins dos testes realizados neste trabalho, até o ponto em que se obtém uma solução onde não há violações das restrições rígidas (nenhum choque de horários). Este limite poderia ser facilmente estendido para outro critério arbitrário em observância às restrições brandas. Porém, como foram replicadas centenas de execuções do código para colher resultados, precisou-se restringir a qualidade das soluções para observar apenas as restrições rígidas e assim viabilizar um tempo de execução menos proibitivo.

5 EXPERIMENTAÇÃO E ANÁLISE DOS RESULTADOS

5.1 ANÁLISE DE PARÂMETROS

Visando testar e validar parâmetros, o algoritmo foi submetido a extensivos testes de performance. Foram testadas três instâncias diferentes de escolas hipotéticas com cinco, seis e oito turmas. Para cada instância, testou-se os parâmetros com taxa 25%, 50% e 75%, tanto de crossover como de mutação.

Para garantir certa rigidez estatística, todos os testes foram repetidos dez vezes sob as mesmas condições e uma média aritmética simples foi calculada para avaliar a performance. A tabela 6 ilustra a média das gerações necessárias para se obter uma solução sob diferentes configurações de parâmetros, enquanto a tabela 7 ilustra os respectivos desvios-padrões de cada configuração.

Observou-se o melhor desempenho para o algoritmo com parâmetros de cruzamento em 50% e de mutação em 75% (menor média necessária e menor desvio-padrão geral).

O desempenho superior do algoritmo observado para uma instância de 6 turmas em relação à instância de 5 turmas se deve ao fato que, em nossa escola hipotética, um professor de português ou matemática precisaria lecionar quatro aulas por semana em cada turma (seguindo as normas padrão para uma escola de ensino fundamental do estado de Santa Catarina). Isso faz com que um mesmo professor dessas disciplinas consiga cobrir, no máximo, 5 turmas. A partir de 6 turmas, o algoritmo automaticamente aloca mais um professor para a matéria referida e divide igualmente a carga horária das turmas

Mutação	25%			50%			75%		
Crossover	25%	50%	75%	25%	50%	75%	25%	50%	75%
5 Turmas	3340	3090	2643	1479	1805	814	853	1188	1247
6 Turmas	1159	1404	1611	1122	960	678	610	667	871
8 Turmas	3974	4390	3675	3113	2726	2396	2240	1514	2054
Média total	2824	2961	2643	1905	1830	1296	1234	1123	1391

Tabela 6 – Tabela das médias de gerações necessárias para se obter uma solução sob cada configuração de parâmetros.

Mutação	25%			50%			75%		
Crossover	25%	50%	75%	25%	50%	75%	25%	50%	75%
5 Turmas	1673	1982	1239	1196	1591	634	604	720	1254
6 Turmas	789	765	1245	559	409	221	213	283	309
8 Turmas	2612	2532	1947	1605	1607	1117	1401	657	1202
Média total	1691	1760	1477	1120	1203	657	739	553	921

Tabela 7 – Tabela dos desvios-padrões referentes a cada configuração de parâmetros.

entre os dois professores, gerando assim uma grade com mais janelas e consequentemente facilitando a obtenção de possíveis soluções por parte do algoritmo genético.

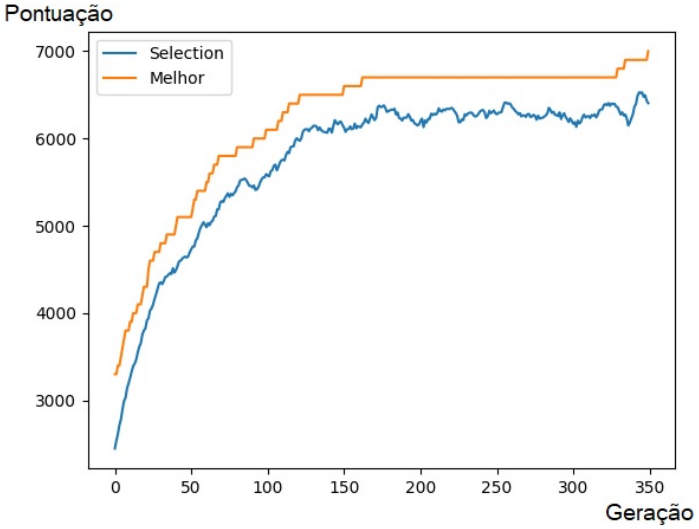

Figura 8 – Gráfico da evolução das populações ao longo das gerações. Em laranja, tem-se o score do melhor indivíduo e, em azul, o score médio da população após a etapa de seleção. O eixo y representa o score e o eixo x, a geração atual.

O gráfico da figura 8 ilustra o comportamento do algoritmo ao longo de uma execução completa. Nota-se uma subida abrupta logo no início, onde uma população inicial com indíviduos relativamente "ruins"é rapidamente substituida por outras que possuem membros mais aptos. O ritmo decresce gradualmente até chegar a um ponto de quase solução, onde ele permanece pela maior parte do tempo. Isto é esperado, visto que a probabilidade de resolver os últimos conflitos existentes em uma grade quase "perfeita"é bem menor do que resolver conflitos em uma grade ruim.

Cabe ressaltar que, devido à escolha de representação e pela forma na qual a mutação é implementada, o problema de máximos locais é inexistente nesta versão de algoritmo genético. A forma lenta e gradual da qual as soluções melhoram ao longo das gerações nunca leva a um estado onde não seja possível obter uma solução definitiva. Isso ocorre pelo fato de que as configurações dos professores (número de aulas por semana e aulas por turma) são sempre mantidas e a mutação apenas troca dois genes de lugar na mesma cadeia.

5.2 TESTES DE VARIAÇÕES

Nesta seção serão testadas técnicas utilizadas na atualidade para melhoria de performance em algoritmos genéticos. Cada técnica terá uma subseção dedicada à sua explicação e resultados. Para os fins desta seção, serão utilizados os parâmetros obtidos pelos testes da seção 3.1, com uma instância de escola com 8 turmas.

5.2.1 Elitismo

A técnica do elitismo consiste em identificar o melhor indivíduo em uma população em um dado momento e automaticamente selecioná-lo para compor a próxima geração (seja com uma ou mais cópias). Na versão base de nosso algoritmo foi implementado o elitismo com oito cópias do melhor indivíduo e, nesta seção, espera-se obter empiricamente qual seria o número ótimo de cópias para uma melhor performance.

A tabela 8 ilustra os testes realizados com o elitismo atribuído a 8, 10 e 15 cópias do melhor indivíduo, respectivamente. Os testes foram repetidos dez vezes para maior rigidez estatística.

Execução	8 indivíduos	10 indivíduos	15 indivíduos
1	1481	1417	4825
2	605	990	622
3	1376	569	581
4	1766	1150	1290
5	2665	1055	3253
6	1119	1733	924
7	919	838	1219
8	2408	1030	1955
9	999	3595	1948
10	1807	792	2216
Média	1514	1316	1883
Desvio	657	864	1318

Tabela 8 – Desempenho do elitismo sob variados parâmetros. Dados expressos em número de gerações.

Nota-se que o melhor resultado foi obtido com 10 cópias do melhor indivíduo, um parâmetro intermediário entre o preenchimento da população com bons indivíduos e boa variabilidade genética provinda de indivíduos com score mais baixo.

5.2.2 Algoritmo Híbrido

Em um estudo datado de 2020, um grupo de pesquisadores (KATOCH; SINGH CHAUHAN; KUMAR, 2020) realizou um levantamento (survey) sobre algoritmos genéticos, onde foi apontado o bom desempenho de uma técnica híbrida em AG's quando aplicada a problemas de agendamentos em geral. Tal técnica consiste em utilizar-se do algoritmo orinal para chegar a um ponto de quase solução (quando a população atinge um fitness médio pré-estabelecido) e então obter um resultado aplicando métodos convencionais de busca (local search).

Foi implementada uma versão de tal algoritmo híbrido para testes de performance executados sobre os parâmetros obtidos nas seções anteriores e com o elitismo de dez

indivíduos. Nesta versão, utiliza-se o algoritmo genético padrão para obter uma solução contendo até dois conflitos de horários para então chamar o método de busca convencional.

Foi testado um método de busca local simples onde identifica-se um conflito de horário existente na grade e troca-se as posições (na mesma linha da matriz) a fim de consertá-lo. Como na nossa instância todas as turmas precisam ter exatamente uma aula alocada em cada horário da semana, isso faz com que em cada coluna da matriz de grade horária o número de cada turma deve aparecer exatamente uma vez. Tal fato facilita o método de busca, visto que, para consertar um conflito, basta procurar a coluna da matriz onde a turma que originou o conflito não possui um horário alocado e realizar a troca de genes.

A figura 9 ilustra uma grade com dois conflitos (circulados em amarelo e verde, respectivamente). O método de busca iria eleger um dos dois números circulados em amarelo e trocá-lo por seu respectivo par circulado em vermelho (a troca precisa obrigatoriamente ser realizada na mesma linha da matriz para garantir o número de aulas que cada professor precisa lecionar).

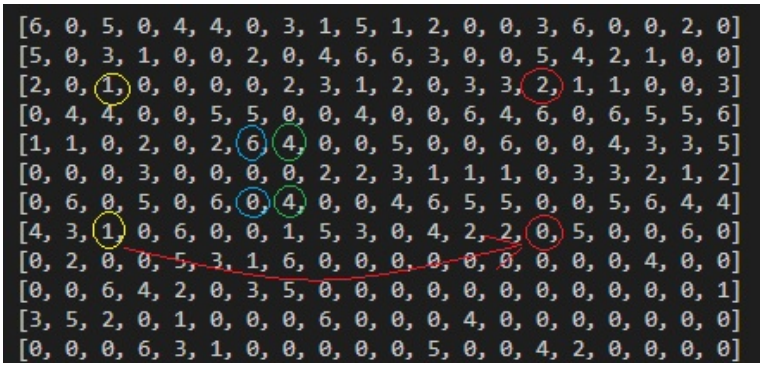

Figura 9 – Ilustração de um possível comportamento do método de busca local.

Foi observado, porém, um problema inerente no que diz respeito ao comportamento do método de busca local e a forma com que ele interage com a instância de grade horária deste trabalho. A simples troca de posições na correção de um conflito, na maioria dos casos, acaba gerando outros conflitos em outras colunas da tabela. Isso acarretou em uma execução infinita da aplicação em todos os testes feitos, nunca convergindo a um resultado ideal. Pode-se notar pela figura 9 que, caso o número 4 circulado em verde fosse trocado pelo número zero (em azul), o problema seria resolvido. Em contrapartida, se o número trocado fosse o 6, isso acarretaria em outro conflito, gerando assim uma reação em cadeia.

Conclui-se, então, que o método de busca local não é adequado para a solução do timetabling problem para esta instância específica que contempla o escopo deste trabalho. Tal resultado vai de encontro com o do trabalho citado na seção 1.3, onde os pesquisadores obtiveram apenas um êxito parcial na aplicação de um algoritmo híbrido, mesmo utilizando-se de um método de busca mais elaborado (REZAEIPANAH; MATOOR; AHMADI, 2020).

5.2.3 Otimização da População Inicial

Nesta seção investigaremos o impacto de melhorias no método de geração da população incial. Em um trabalho datado de 2006, um grupo de pesquisadores finlandeses realizou uma pesquisa acerca de diferentes métodos de geração populacional (MAARANEN; MIETTINEN; PENTTINEN, 2006). Foram testados métodos de geração pseudo-aleatórios e quasi-aleatórios. Concluiu-se que tais métodos possuem um impacto considerável na execução e obtenção de uma solução rápida. A seguir, vericaremos se os resultados também se aplicam à nossa instância de problema.

Primeiramente foi implementado um método gerador com o auxílio de um contador, que verifica a quantidade de conflitos existentes em uma mesma coluna da grade antes de alocar uma turma. Caso hajam mais do que três conflitos, o algoritmo tenta gerar outro número aleatório, até um máximo de 10 tentativas. Caso nenhuma tentativa seja bem-sucedida, é alocado um número aleatório (respeitando a quantidade de aulas que cada professor precisa lecionar), mesmo se isso acarretar em um conflito.

Verificou-se uma melhora considerável no score das populações iniciais. Os resultados de dez execuções do algoritmo com uma instância de 8 turmas são apresentados na tabela 9 (segunda coluna), onde observa-se um desempenho semelhante ao do algoritmo original, porém com um desvio padrão menos acentuado. Isso nos diz que, apesar de não possuir tanto impacto no número de gerações necessárias para se obter uma resposta, esta técnica viabiliza tempo de execução mais constante e menos sujeito à aleatoriedade do AG.

Visto que cada professor obrigatoriamente dará ao menos uma aula em cada turma, testou-se também uma variação de inicialização onde primeiramente popula-se as diagonais da matriz com números crescentes, com o intuito de alocar turmas de forma a garantir menos colisões. A figura 10 ilustra como seria a inicialização da matriz de grades horários.

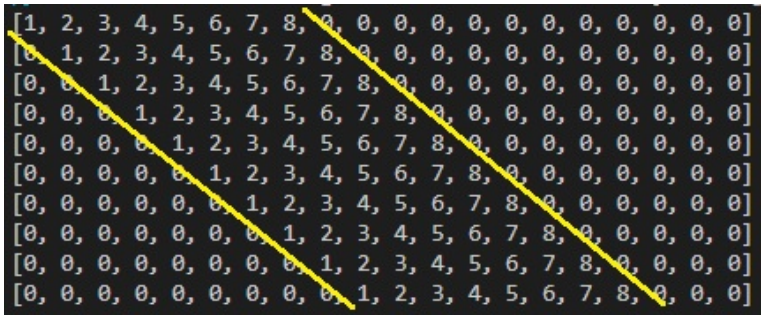

Figura 10 – Ilustração da matriz inicial com suas diagonais preenchidas.

Com o preenchimento inicial realizado, a grade é então submetida ao processo comum de inicialização pelo método pseudo-aleatório padrão.

Observa-se pela tabela 9 (terceira coluna) que o método possui desempenho semelhante aos outros, com a especifidade de apresentar um desvio-padrão muito elevado.

Isso significa que , da mesma forma que pode-se gerar uma resposta com pouquíssimas gerações, pode-se também levar um tempo muito maior para se alcançar a condição de parada.

Execução	Original	Contador	Diagonal
1	1417	1067	2268
2	990	978	859
3	569	1149	848
4	1150	903	2533
5	1055	2683	1072
6	1733	1729	659
7	838	1361	1966
8	1030	665	1292
9	3595	1502	1790
10	792	1211	315
Média	1316	1324	1360
Desvio	864	657	1318

Tabela 9 – Desempenho das duas técnicas empregadas em comparação ao modelo original. Resultados expressos em número de gerações. Testes executados com elitismo de 10 indivíduos e uma instância de 8 turmas.

6 CONCLUSÕES E TRABALHOS FUTUROS

O algoritmo genético cumpriu seu objetivo principal de resolver um problema combinatorial de alta complexidade em um tempo razoável de execução. Uma solução que satisfaz as restrições rígidas foi produzida em todos os testes. Tal solução, ainda que não seja a ideal, seria satisfatória para um colégio de porte pequeno.

Ao longo do tempo, várias tentativas de resolver o timetabling problem via AG's foram realizadas, com diversas inovações e ideias sendo propostas em cada abordagem. De tudo, o aspecto que parece ter um impacto maior na performance do algoritmo é a representação em si. Neste trabalho, optamos por uma representação simples porém completa de professores, turmas e suas grade-horárias. Tal representação exigiu certas adaptações de algumas partes do algoritmo original, mas, em suma, ela atendeu aos objetivos esperados do problema.

Verificamos experimentalmente que a geração de uma população inicial melhor não necessariamente acarreta em um menor tempo de execução para o programa. A complexidade parece estar mais atrelada ao final da execução do algoritmo, onde instâncias quase solucionadas estão presentes na população, e a chance de resolver-se os conflitos restantes é muito baixa.

Nossa função de aptidão, ainda que simples, cumpriu seu papel de separar bons indivíduos dos mais fracos, favorecendo instâncias que possuem menos conflitos, permitindo assim uma convergência gradual e satisfatória para uma solução.

Com testes preliminares, verificou-se que o método da roleta era o mais adequado para nosso programa. A simplicidade desta técnica permite um bom tempo de execução ao mesmo tempo que economiza recursos (memória) se comparados a outros métodos, como o do torneio, por exemplo.

Ainda sobre a etapa de seleção, o estudo do elitismo mostrou que um balanço entre dominância genética por parte do indivíduo e variabilidade é o ideal para se alcançar uma boa solução no menor número de gerações possível. O fator de dez indivíduos estarem presentes na próxima geração, com nove sendo aptos para crossover e mutação foi importante para melhorar o score de gerações futuras e ao mesmo tempo garantir a permanência do indivíduo dominante na população.

A abordagem para o crossover realizada neste trabalho possibilitou uma maior rapidez por parte do algoritmo ao economizar comparações e gerações de números aleatórios, ao passo que ainda permitiu boa variabilidade genética.

A mutação foi um dos grandes desafios para a adaptação do AG à nossa representação. Desenvolveu-se esta etapa de tal forma que, ao mesmo tempo que poupa tempo de execução ao realizar menos iterações, também mantém as restrições impostas de carga-horária e aulas por turma, aumentando a variabilidade genética das soluções e garantindo boa convergência.

Testes extensivos possibilitaram a obtenção de parâmetros ideais para as taxas de mutação e crossover, a fim de otimizar o número de gerações necessárias para resolver-se o problema. Pelos resultados, conclui-se que, para a instância do problema em questão, a variabilidade genética deve ser bastante favorecida, ainda mais tendo em vista que o problema de máximos locais é inexistente nesta representação.

O teste de um algoritmo híbrido falhou ao prover uma solução adequada ao problema em um tempo não-proibitivo. Especificidades da implementação fizeram com que o algoritmo de busca local entrasse em execução infinita e nunca retornasse uma resposta. Cabe, porém, uma exploração mais aprofundada em torno desta temática. Novas tentativas de otimizar o algoritmo de busca de forma a driblar os problemas encontrados podem ser promissoras.

Em síntese, os objetivos específicos propostos por este trabalho foram atendidos de forma satisfatória. Uma aplicação para gerar grades de horários de professores e turmas de escolas do ensino fundamental de forma a não haver conflitos foi desenvolvido com sucesso. Testes extensivos foram realizados para medir e calibrar bons parâmetros de execução. E por, último, as três técnicas mais comuns de otimização de AG's foram implementadas e seus resultados foram testados e analizados neste documento.

6.1 TRABALHOS FUTUROS

Diante do resultado positivo obtido ao medir-se a performance do algoritmo, prova-se que a heurística de AG's é bastante adequada para resolver problemas de tabulações de horários. Com isso, um passo adiante seria adaptar o programa desenvolvido neste trabalho para uma escola real de ensino fundamental ou médio de Santa Catarina.

Tal projeto requeriria mudar certos aspectos do algoritmo atual, tais como reusabilidade e opções individuais de cada professor. Como problemas reais são sempre mais complexos que instâncias reproduzidas em laboratório, seria necessário entrar em contato direto com uma escola e estudar a fundo as necessidades e restrições da organização interna.

Além disso, um passo relativamente importante para viabilizar a execução do algoritmo em escolas maiores seria a exploração do paralelismo. Tal aspecto não foi implementado neste trabalho pela simplicidade das instâncias de testes, mas, como descrito anteriormente neste documento, o paralelismo tem enorme potencial de melhorar a performance de um AG devido à sua natureza inerentemente independente em certas etapas da execução.

REFERÊNCIAS

ALBA, Enrique; M. TROYA, José. A Survey of Parallel Distributed Genetic Algorithms.

BULLINARIA, JOHN A., Sanchez-Velazco, Jose. Gendered Selection Strategies in Genetic Algorithms for Optimization, 2003.

BURKE, Edmund; ELLIMAN, David; WEARE, Rupert. A Genetic Algorithm for University Timetabling. **AISB Workshop on Evolutionary Computation**, workshop notes, 1994.

CORLONE, Alberto; DORIGO, Marco; MANIEZZO, Vittorio. Genetic Algorithms and Highly Constrained Problems, The Timetabling Case. **Parallel Problem Solving From Nature**, v. 6, p. 55–59, 1990.

D. BETHKE, Albert. Comparison of Genetic Algorithms and Gradient-Based Optimizers on Parallel Processors : Efficiency of Use of Processing Capacity. **Logic of Computers Group Technical Report**, n. 197, 1976.

DEB, K. A fast and elitist multiobjective genetic algorithm: NSGA-II. **IEEE Transactions on Evolutionary Computation**, v. 6, n. 2, p. 182–197, 2002.

E. HOPCROFT, JOHN; D. ULLMAN, JEFFREY. **Formal Languages and their Relation to Automata**. 1. ed. Cambridge: Addison-Wesley Publishing Company, Inc., 1969.

FANG, Hsiao-Lan. Genetic Algorithms in Timetabling and Scheduling, 1994.

GAREY, Michael R.; JOHNSON, David S. **Computers and Intractability; A Guide to the Theory of NP-Completeness**. USA: W. H. Freeman Co., 1990. ISBN 0716710455.

GONSALVES, T.; OISHI, R. Artificial immune algorithm for exams timetable. **J. Inf. Sci. Comput. Technol.**, n. 4, p. 287–296, 2015.

KATOCH, Sourabh; SINGH CHAUHAN, Sumit; KUMAR, Vijay. A review on genetic algorithm: past, present, and future, 2020.

LIPOWSKI, Adam; LIPOWSKA, Dorota. Roulette-wheel selection via stochastic acceptance. **Physica A: Statistical Mechanics and its Applications**, v. 391, 2011.

MAARANEN, Heikki; MIETTINEN, Kaisa; PENTTINEN, Antti. On initial populations of a genetic algorithm for continuous optimization problems. **J Glob Optim**, v. 37, p. 405–436, 2006.

MELANIE, Mitchell. **An Introduction to Genetic Algorithms**. 5. ed. Cambridge: Massachusetts Institute of Technology, 1999.

PAECHTER, B. *et al.* Two solutions to the general timetabling problem using evolutionary methods. **Proceeding of the First IEEE Conference on Evolutionary Computation**, p. 300–305, 1994.

POTVIN, Jean-Yves. Genetic algorithms for the traveling salesman problem. **Annals of Operations Research**, p. 337–370, jun. 1996.

REZAEIPANAH, Amin; MATOOR, Samaneh Sechin; AHMADI, Gholamreza. A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search. **Applied Intelligence**, v. 51, p. 467–492, 2020.

SASTRY, K.; D.E., Goldberg; KENDALL, G. **Genetic Algorithms In Search Methodologies**. [*S.l.*: *s.n.*], 2014.

SHUKLA, Anupriya; PANDEY, Hari; MEHROTRA, Deepti. Comparative Review of Selection Techniques in Genetic Algorithm. **2015 1st International Conference on Futuristic Trends in Computational Analysis and Knowledge Management, ABLAZE 2015**, fev. 2015. DOI: 10.1109/ABLAZE.2015.7154916.

TIERNEY, Kevin. A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms, 2009.

ULLMAN, J. D. Polynomial complete scheduling problems. **ACM SIGOPS Operating Systems Review**, v. 7, n. 4, p. 96–101, 1973.

ZHONG, J.H. *et al.* A differential evolution algorithm with dual populations for solving periodic railway timetable scheduling problem. **IEEE. Trans. Evol. Comput.**, v. 17, p. 512–527, 2013.

APÊNDICE A – ALGORITMO

A seguir, é apresentado o código-fonte do algoritmo escrito em python, suas classes e seus métodos. O mesmo também se encontra em um repositório público do GitHub que pode ser acessado pelo link: https://github.com/Gagocampos/GeneticAlgorithm

A.1 INDIVIDUO.PY

```
from Professor import Professor
import copy

class Individuo:
    def __init__(self, professores) -> None:
        self.aptidao = 0
        self.professores = copy.deepcopy(professores)

    # Calculo de aptidao do individuo
    def fitnessEvaluation(self, turmas):
        self.aptidao = 7000
        for i in range(len(self.professores[0].grade)):
            count = [0 for i in range(turmas)]
            for j in range(len(self.professores)):
                num = self.professores[j].grade[i]
                if num > 0:
                    count[num - 1] += 1
            for posicao in count:
                self.aptidao -= abs((posicao - 1) * 50)
```

A.2 PROFESSOR.PY

```
class Professor:
    def __init__(self, nome, disciplina, aulas_por_turma):
        self.nome = nome
        self.disciplina = disciplina
        self.aulas_por_turma = aulas_por_turma
        self.grade = []
```

A.3 MAIN.PY

```
from cProfile import label
from Professor import Professor
from Individuo import Individuo
import random
import matplotlib.pyplot as plt
import copy

# parametros globais
solucao = []
solucao_fitness = 0
solucao_index = 0
condicao_de_parada = True
generation = 0
media_selection, media_crossover, media_mutation, melhores = [], [], [],


# Retorna o score maximo e minimo
def max_min(populacao):
    menor, maior = 100000,0
    for individuo in populacao:
        value = individuo.aptidao
        if value > maior:
            maior = value
        elif value < menor:
            menor = value
    return menor, maior


def calcular_media(populacao, vetor):
    total = 0
    for individuo in populacao:
        total += individuo.aptidao
    vetor.append(total/len(populacao))


def buscar_melhor_ind(populacao):
    max = 0
    for n in range(len(populacao)):
        aux = populacao[n].aptidao
```

```
        if aux > max:
            max = aux
            solucao_index = n
    return solucao_index, max


# Metodo de selecao (lembrar do janelamento)
def selection(populacao):
    nova_populacao = []
    soma_total = 0
    probabilidade = []

    for individuo in populacao:
        soma_total += individuo.aptidao

    for individuo in populacao:
        probabilidade.append(individuo.aptidao/soma_total)

    index, temp = buscar_melhor_ind(populacao)
    for n in range(0,len(populacao)):
        if n < 10:
            nova_populacao.append(copy.deepcopy(populacao[index]))
        else:
            aux = 0
            num = random.uniform(0,1)
            for i in range(0,len(probabilidade)):
                aux += probabilidade[i]
                if num <= aux:
                    # print("Individuo selecionado: ", i)
                    nova_populacao.append(copy.deepcopy(populacao[i]))
                    break
    del populacao
    del probabilidade
    return nova_populacao


# Metodo que realiza a troca de genes
def crossover(populacao):
    for index, individuo in enumerate(populacao):
```

```
        if index > 0:
            num = random.uniform(0,1)
            if num > 0.5:
                parceiro = int(random.uniform(0,len(populacao)-1))
                for n in range(0, int(len(individuo.professores)/2)):
                    aux = copy.deepcopy(individuo.professores[n].grade)
                    individuo.professores[n].grade = copy.deepcopy(popula
                    populacao[parceiro].professores[n].grade = copy.deepc
                    del aux


# Metodo que realiza a mutacao
def mutation(populacao):
    for index, individuo in enumerate(populacao):
        if index > 0:
            rng = random.uniform(0,1)
            if rng > 0.25:
                professor = individuo.professores[int(random.uniform(0,le
                horario1 = int(random.uniform(0,len(professor.grade)))
                horario2 = int(random.uniform(0,len(professor.grade)))
                aux = professor.grade[horario1]
                professor.grade[horario1] = professor.grade[horario2]
                professor.grade[horario2] = aux
                del aux


# Dados de entrada
turmas = 8
aulas_por_dia = 4
populacao = []
professores = []


# metodo que torna o algoritmo hibrido
def solve(solucao):
    counter = []
    for n in range(len(solucao.professores[0].grade)):
        count = [0 for i in range(turmas)]
        counter.append(count)
```

```
    for n in range(len(solucao.professores[0].grade)):
        for m in range(len(solucao.professores)):
            if solucao.professores[m].grade[n] != 0:
                counter[n][solucao.professores[m].grade[n] - 1] += 1
    for n in range(len(counter)):
        for m in range(len(counter[n])):
            if counter[n][m] > 1:
                for i in range(len(counter)):
                    if counter[i][m] == 0:
                        for r in range(len(solucao.professores)):
                            if solucao.professores[r].grade[n] == m+1:
                                aux = solucao.professores[r].grade[n]
                                solucao.professores[r].grade[n] = solucao
                                solucao.professores[r].grade[i] = aux
                                break
                        break
    del(count)

# Inicializacao da populacao
for n in range(0,100):

    professores.append(Professor("Natan", "Geografia",2))
    professores.append(Professor("Emerson", "Historia",2))
    professores.append(Professor("Camila", "Portugues",4))
    professores.append(Professor("Jorge", "Portugues",4))
    professores.append(Professor("Clara", "Ciencias",2))
    professores.append(Professor("Carlos", "Educacao Fisica",2))
    professores.append(Professor("Maria", "Artes",1))
    professores.append(Professor("Ana", "Ensino Religioso",1))
    professores.append(Professor("Evelyn", "Matematica",4))
    professores.append(Professor("Marcondes", "Matematica",4))
    professores.append(Professor("Marta", "Ingles",1))
    professores.append(Professor("Alenka", "Espanhol",1))

    offset = 0
    for professor in professores:
        limit = 0
        if professor.aulas_por_turma * turmas > aulas_por_dia * 5:
            limit = int(turmas / 2)
```

```
            else:
                limit = turmas
            janelas = (aulas_por_dia * 5) -
            (professor.aulas_por_turma * limit)
            professor.grade = [-1 for i in range(aulas_por_dia * 5)]
            count = 1
            for i in range(offset, limit + offset):
                professor.grade[i] = count
                count += 1
            for n in range((aulas_por_dia * 5)):
                if n < offset or n >= limit + offset:
                    while(True):
                        num = random.randint(0,limit)
                        if num !=0 and professor.grade.count(num) < professor
                            professor.grade[n] = num
                            break
                        elif num == 0 and professor.grade.count(num) < janela
                            professor.grade[n] = num
                            break
            offset += 1

        discp = ""
        for n in range(len(professores)):
            if n == 0:
                discp = professores[0].disciplina
            else:
                if professores[n].disciplina == discp:
                    for m in range(0, (aulas_por_dia * 5)):
                        if professores[n].grade[m] != 0:
                            professores[n].grade[m] += int(turmas/2)
                discp = professores[n].disciplina

        populacao.append(Individuo(professores))
        professores.clear()

    for professor in populacao[0].professores:
        print(professor.grade)
    print("----------------------------------------")
```

```
for individuo in populacao:
    individuo.fitnessEvaluation(turmas)
while generation < 7000:
    populacao_aux = selection(populacao)
    populacao.clear()
    populacao = copy.deepcopy(populacao_aux)
    populacao_aux.clear()
    calcular_media(populacao, media_selection)

    crossover(populacao)

    mutation(populacao)

    for individuo in populacao:
        individuo.fitnessEvaluation(turmas)
    generation += 1
    r, value = buscar_melhor_ind(populacao)
    melhores.append(value)
    if value > solucao_fitness:
        print(value)
        solucao = copy.deepcopy(populacao[r])
        solucao_fitness = value
        if solucao_fitness == 7000:
            break

for professor in solucao.professores:
    print(professor.grade)
print(generation)

x = []
for n in range(len(media_selection)):
    x.append(n)

plt.plot(x, media_selection, label = "Selection")
plt.plot(x, melhores, label = "Melhor")
plt.legend()
plt.show()
```

APÊNDICE B – ARTIGO

Algoritmos Genéticos Aplicados ao Problema de Alocação de Horários e Professores em Escolas do Ensino Fundamental

Gabriel Campos Albino[1]

[1]Instituto de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

gcampos1337@gmail.com

Abstract. *This work aims to help the community by proposing a solution to the problem of scheduling subjects and time slots, which is recurrent in any and all educational institutions. A performance measurement of a Genetic Algorithm adapted to the famous timetabling problem (NP-Complete) was carried out, aiming to empirically establish the ideal execution parameters and obtain a case study of the implementation of several modern techniques applied over the original algorithm.*

Resumo. *Este trabalho tem por objetivo auxiliar a comunidade ao propor uma solução ao problema de agendamento de disciplinas e horários, que é recorrente em toda e qualquer instituição de ensino. Foi realizada uma medição de desempenho de um Algoritmo Genético adaptado ao famoso timetabling problem (NP-Completo), visando estabelecer empiricamente os parâmetros ideais de execução e obter um estudo de caso da implementação de variadas técnicas modernas implementadas sobre o algoritmo original.*

1. Introdução

Em todo início de um novo período letivo, as instituições de ensino se deparam com o problema de organizar sua grade de horários de forma que não hajam conflitos entre disciplinas de um mesmo período ou choques de disponibilidade na agenda de alunos e professores. Vários requisitos precisam ser atendidos, visto que a grade dos alunos devem sempre estar completas e, ao mesmo tempo, devem satisfazer eventuais preferências ou limitações de agenda por parte dos professores. O fato de tal problema ser reconhecidamente NP-Completo [Ullman 1973] torna a obtenção de uma solução exata e ótima uma tarefa praticamente inviável.

Tendo em vista tais dificuldades, a aplicação de uma heurística provinda da inteligência artificial popularmente conhecida como algoritmos genéticos pareceu adequada para encontrar uma solução aproximada para o problema. Tais algoritmos são amplamente utilizados nas mais diversas áreas do conhecimento e de aplicação, justamente por oferecerem uma solução aproximada e satisfatória para problemas combinatoriais de complexidade elevada [Melanie 1999], tais como obter solução de polinômios de alto grau ou o problema do caixeiro viajante.

Ao final deste trabalho, espera-se obter uma análise extensa do desempenho de um Algoritmo Genético adaptado ao problema de tabulação de horários, assim como estimar os parâmetros otimizados para sua execução e explorar técnicas de melhoria de desempenho.

2. Timetabling Problem

O problema do encaixe de horários (do inglês, Timetabling Problem) diz respeito a eventos (aulas, exames, etc) que devem ser arranjados em um número de slots, sujeitos a diversas limitações. A necessidade de métodos poderosos e eficientes fica clara ao observar que, para um número x de eventos a serem encaixados em um numero t de slots de tempo, temos x^t possíveis tabulações candidatas a solucionar o problema.

As abordagens mais comuns para a solução de tais problemas consistem simplesmente em achar a tabulação mais curta que satisfaça as restrições impostas utilizando um algoritmo de coloração de grafos [Fang 1994]. Tais algoritmos podem ser executados em tempo polinomial em uma pequena minoria de casos (coloração com duas cores), mas, em sua vasta maioria, são classificados como NP-Completos [Garey and Johnson 1990].

3. Algoritmos Genéticos

Algoritmos Genéticos podem geralmente ser subdivididos em duas etapas principais: geração da população inicial e evolução. Em sua primeira etapa, as potenciais soluções iniciais do domínio do problema são codificadas em representações que devem suportar as variações e operações de seleção necessárias; geralmente, essas representações são tão simples quanto uma cadeia de bits, apesar de haverem representações mais complexas para problemas específicos.

Assim como na biologia natural, as entidades presentes em um algoritmo genético seguem uma hierarquia: um gene diz respeito à uma característica específica - um genótipo - e o conjunto de genes forma um cromossomo (ou, no caso da analogia, um indíviduo). Cada indivíduo é formado por características únicas que serão avaliadas posteriormente, e o conjunto de indivíduos é denominado população. O objetivo final do algoritmo é prover uma população que contenha ao menos um indivíduo que seja apto o suficiente de acordo com algum parâmetro pré-estabelecido, avaliado através de seus genes.

Na segunda etapa, os algoritmos de acasalamento e mutação, análogos à atividade reprodutiva de formas vivas biológicas, produzem uma nova geração de indivíduos que recombinam o material genético de seus pais. Para isso, primeiro faz-se uma análise dos indivíduos (de acordo com uma função de aptidão adequada ao domínio do problema) que posteriormente poderão ser escolhidos para o acasalamento, com probabilidade de escolha proporcional à sua aptidão.

A fase de "acasalamento" consiste na escolha de um operador genético para realizar a troca de "genes". O operador mais comum é a recombinação (ou crossover), que basicamente toma duas soluções candidatas e as divide, seja exatamente na metade ou em algum ponto arbitrário, originando dois filhos, cada um contendo sua primeira metade de genes vinda de um genitor e outra metade vinda do outro. A figura 1 ilustra uma possível aplicação deste operador genético (a imagem é meramente ilustrativa e não reflete a representação que será utilizada neste trabalho).

Durante a segunda etapa, existe ainda a possibilidade de ocorrer uma "mutação", que é um operador genético que toma um único candidato e troca de forma aleatória algum de seus aspectos. Este operador é de suma importância, visto que uma população inicial pode não conter um componente essencial da solução.

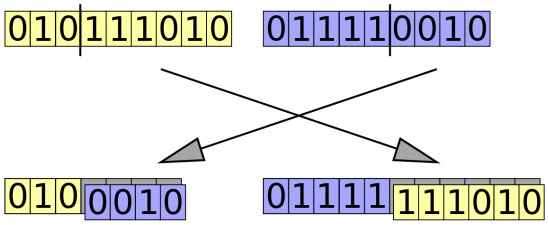

Figure 1. Exemplo de uma aplicação hipotética do crossover

A segunda etapa do algoritmo irá então se repetir até que algum critério de parada pré-estabelecido seja alcançado (como, por exemplo, se a função de aptidão alcançar certo limiar).

4. Trabalhos Relacionados

Utilizar algoritmos genéticos para a solução do problema de tabulação de horários não é algo novo. As primeiras tentativas datam de 1990, quando um grupo de pesquisadores [Corlone et al. 1990] construiu uma tabulação para as aulas de uma escola secundária italiana usando a abordagem de AG's e reportaram a primeira aplicação desse tipo a ser bem-sucedida. A representação escolhida era formada por uma quíntupla, consistida por um recurso (professor), intervalos de tempo (horas), aulas, uma matriz (a tabulação em si) e uma função a ser maximizada. Cada linha da matriz representava um professor, cada coluna era um horário e em cada célula era alocada uma aula. O resultado foi uma aplicação que satisfazia todas as limitações obrigatórias, enquanto minimizava a violação das limitações secundárias e opcionais.

Em 1994, surgiu a ideia inovadora de usar um algoritmo de coloração de grafos para primeiramente criar uma tabulação factível que satisfizesse todas as condições obrigatórias (que eram: um estudante não pode realizar dois exames ao mesmo tempo e as salas de aula devem acomodar todas os estudantes que realizariam a prova), para somente então operar um algoritmo genético sobre a tabulação resultante [Burke et al. 1994]. Este trabalho deu origem à uma aplicação de desempenho satisfatório, mas acarretou em uma complexidade muito grande na escolha de indivíduos para o crossover e um tratamento a mais precisou ser feito nos casos em que a tabulação se tornava não-factível.

Também em 1994, investigou-se o uso de representações indiretas para o timetabling problem. Nesta abordagem, os cromossomos codificam instruções para a construção de uma tabulação, ao invés de representar a entidade em si. A ideia geral é que os cromossomos armazenem informações sobre qual evento agendar em seguida, e onde alocá-lo. Caso tal evento não possa ser alocado sem violar uma condição obrigatória, o cromossomo codificaria onde procurar por um espaço alocável e factível [Paechter et al. 1994]. A aplicação resultante tende a trabalhar com um espaço de busca muito menos esparso que o normal, mas, consequentemente, isso leva à possibilidade de uma provável resposta ótima nunca ser encontrada.

Mais recentemente, um grupo de pesquisadores propôs uma abordagem híbrida única para a solução do timetabling problem em universidades [Rezaeipanah et al. 2020]. A versão foi denominada IPGALS (do inglês, Improved Parallel Genetic Algorithm and

Local Search) e combina paralelismo, algoritmos genéticos e métodos de busca local para minimizar o tempo de execução do programa. A inovação desta proposta reside na forma da qual a busca local foi implementada, atuando logo após a fase de acasalamento com o intuito de melhorar o genes, reintroduzindo possíveis traços genéticos perdidos ou pouco usados e resolvendo conflitos em meio à execução padrão do algoritmo genético. Os resultados reportados foram satisfatórios para instâncias pequenas do problema, mas o IPGALS não foi capaz de gerar soluções factíveis para instâncias maiores.

5. A Abordagem Eleita

A seguir serão descritas em detalhes cada etapa do algoritmo, assim como especifidades sobre a representação, restrições e cálculo de aptidão.

5.1. Restrições

Em relação as restrições, deverão ser obrigatoriamente notadas (restrições rígidas):

1. O mesmo aluno não pode assistir a duas aulas ao mesmo tempo
2. O mesmo professor não pode ministrar duas aulas ao mesmo tempo
3. O número de aulas deve menor ou igual ao número de slots disponíveis

Existem ainda aspectos que são desejáveis, mas não obrigatórios (restrições brandas), tais como:

1. Duas aulas da mesma disciplina não devem acontecer imediatamente uma após a outra para uma mesma turma
2. Aulas de educação física devem ser evitadas em horários logo em seguida às refeições

5.2. Representação

Para o uso correto de um AG aplicado a qualquer problema, o primeiro passo importante é eleger uma representação adequada da entidade em forma de cromossomo. Neste trabalho, utilizaremos uma representação que consiste em codificar cada professor da instituição de ensino em uma longa cadeia de números inteiros, onde cada posição da mesma representaria um horário de aula e cada número inteiro corresponderia a uma turma específica. Como em escolas do ensino fundamental o mais comum é haverem quatro aulas por turno, o horário semanal de um determinado professor que só ensina no período da manhã seria traduzido em uma string com 20 posições (5 dias da semana).

Tal representação foi inspirada no trabalho de pesquisadores italianos citados anteriormente [Corlone et al. 1990], visando manter a simplicidade (a título de performance) e ao mesmo tempo atender todas as necessidades que uma grade-horária de ensino fundamental requer.

Na figura 2, onde observamos as cadeias de 20 inteiros referentes a dois possíveis professores, podemos notar que o número quatro aparece na quarta posição em ambos os casos. Isso significa que a turma 4 foi atribuída a dois professores distintos ao mesmo tempo (no quarto horário da semana, ou seja, no quarto horário da segunda-feira). Isso violaria uma restrição rígida do nosso algoritmo e esta solução receberia uma penalidade elevada na etapa de cálculo de aptidão.

Ana = [28540156463112346751]

Jorge = [37843645871321437615]

Figure 2. Representação dos horários de dois professores. Os 0's representam janelas de tempo livre na grade.

5.3. Geração da População Inicial

A população inicial de indivíduos neste trabalho foi gerada (antes das adaptações estudas) de forma completamente aleatória. Para cada posição da matriz de horários, gera-se um número inteiro aleatório respeitando o intervalo $[0, n]$, onde n é o número de turmas da escola. Daí, verifica-se com o auxílio de um contador se o professor já atendeu o número de aulas por semana para aquela turma sorteada. Se sim, repete-se o processo até sortear uma turma que ainda não foi atendida. Se não, o horário é alocada para aquela determinada turma com sucesso.

Caso o número zero seja sorteado, antes de alocá-lo verifica-se se o professor já atingiu seu número máximo de janelas por semana, que é calculado por $Slots - Turmas \times Aulas/Turma$.

5.4. Função de Aptidão

O cálculo da função de aptidão consiste em uma varredura da matriz de horários de um indivíduo, coluna por coluna. Primeiramente, atribui-se a um indivíduo a pontuação máxima (calculada por $n \times 1000$, onde n é o número de turmas) para que então o score seja reduzido a cada conflito identificado ou restrição não observada. A pontuação máxima deve ser alterada em relação ao número de turmas para evitar possíveis divisões por zero no decorrer da execução do algoritmo.

A penalidade para quebra de restrições rígidas é de 50 para cada repetição da mesma turma em um slot de tempo. Se uma turma for alocada apenas uma vez em um slot da semana, o indivíduo não é penalizado. Já se tal turma for alocada duas vezes, a penalidade será de $2 \times 50 = 100$. Caso ela seja alocada três vezes a penalidade será de $3 \times 50 = 150$, e assim por diante.

A função conta com o auxílio de um vetor contador que possibilita alcançar uma execução com complexidade $O(N \times M)$, sem precisar percorrer a mesma coluna da matriz diversas vezes.

5.5. Seleção

A seleção dos indívíduos aptos para a próxima geração é um fator muito importante para a eficiência e garantia de convergência do algoritmo. Para este trabalho, o método de seleção escolhido foi o da roleta. Este método se provou eficiente e nos permitiu obter resultados satisfatórios para a abordagem eleita.

O método do torneio também foi implementado e seus resultados foram testados de forma preliminar. Tal método, porém, pouco influenciou no tempo de execução total

do algoritmo e, por isso, voltamos ao método da roleta original que havia sido eleito para esta etapa, por questão de maior simplicidade.

Foi implementado um elitismo de dez indivíduos. Isso significa dizer que, no começo da fase de seleção, identifica-se o melhor indivíduo da geração anterior e automaticamente adiciona-se dez cópias do mesmo à geração atual. Tal técnica se prova importante para obter-se uma convergência mais acentuada para o algoritmo.

5.6. Crossover

Neste trabalho, o crossover é realizado de tal maneira que metade dos genes de um indivíduo são trocados com os de outro. Ou seja, metade dos professores que compõem uma possível solução são trocados. Essa forma de cruzamento nos proporcionou uma boa variabilidade genética e permitiu que a convergência ocorresse de forma gradual, porém rápida. Nesta fase, dos dez indivíduos que vêm da geração anterior por meio do elitismo, apenas nove ficam aptos a trocarem seus genes (para garantir que a melhor solução não será perdida no decorrer da execução do algoritmo).

Primeiramente, sorteia-se um número aleatório para cada indivíduo seguindo uma distribuição uniforme de 0 a 1 e, caso este número seja maior que 0.5 (parâmetro obtido experimentalmente), o indivíduo é selecionado para o "acasalamento". Então, sorteia-se um número inteiro no intervalo de 2 a 100 (que é o tamanho da população, excluindo-se o primeiro indivíduo para manter-se o elitismo) para representar o indivíduo escolhido como par na troca de genes. Daí então é feita a troca, subtituindo a primeira metade da grade horária de um pela outra metado do outro.

5.7. Mutação

A mutação em neste algoritmmo se comporta de forma diferente da padrão para algoritmos genéticos. Não seria possível trocar aleatoriamente o valor de um gene, pois isso iria desfazer a configuração necessária do número de aulas que cada professor precisa lecionar e certamente algumas turmas não seriam atendidas. Ao invés disso, os genes são simplesmente trocados de posição na mesma linha da matriz de grade de horários. Assim como na etapa de crossover, apenas nove dos dez indivíduos provindos do elitismo têm a possibilidade de sofrer mutação, para preservar a melhor solução.

Para a realização da mutação, primeiramente sorteia-se um número aleatório para cada indivíduo da população seguindo uma distribuição uniforme entre 0 e 1. Caso o número seja maior que 0.25, o indivíduo é selecionado para mutar. Então, três números inteiros são sorteados para cada indivíduo selecionado: um inteiro no intervalo de $[0, n-1]$ (onde n é o número de professores da escola) para selecionar qual professor será mutado, e dois inteiros no intervalo de $[0, 19]$ (número de slots na semana), para eleger os slots que serão trocados de lugar.

Desta forma, mesmo com a taxa aparentemente bastante elevada (75%), apenas pouquíssimos genes trocam de fato seus valores na fase de mutação. Essa adaptação se fez necessária para cumprir às restrições de ordem e número de aulas que cada professor deve lecionar. Fazendo-se uma breve comparação com um processo de mutação convencional, onde a taxa é aplicada para cada gene que compõe a solução (e não ao indivíduo como um todo, como é o caso deste trabalho), teríamos uma taxa equivalente descrita pela equação 1 (onde n é o número de professores na escola):

Mutação	25%			50%			75%		
Crossover	25%	50%	75%	25%	50%	75%	25%	50%	75%
5 Turmas	3340	3090	2643	1479	1805	814	853	1188	1247
6 Turmas	1159	1404	1611	1122	960	678	610	667	871
8 Turmas	3974	4390	3675	3113	2726	2396	2240	1514	2054
Média total	2824	2961	2643	1905	1830	1296	1234	1123	1391

Table 1. Tabela das médias de gerações necessárias para se obter uma solução sob cada configuração de parâmetros.

$$T_m = 0,75 \cdot \frac{2}{20 \cdot n} \tag{1}$$

A equação representa a taxa de 75% aplicada sobre dois do total de genes contidos em um individuo. Então, para uma escola com 12 professores (necessário para atender de 6 a 10 turmas), a taxa de mutação equivalente, se considerássemos a chance individual de cada gene, seria de 0,625%.

5.8. Critério de Parada

O algoritmo é executado, para os fins dos testes realizados neste trabalho, até o ponto em que se obtém uma solução onde não há violações das restrições rígidas (nenhum choque de horários). Este limite poderia ser facilmente estendido para outro critério arbitrário em observância às restrições brandas. Porém, como foram replicadas centenas de execuções do código para colher resultados, precisou-se restringir a qualidade das soluções para observar apenas as restrições rígidas e assim viabilizar um tempo de execução menos proibitivo.

6. Análise de Parâmetros

Visando testar e validar parâmetros, o algoritmo foi submetido a extensivos testes de performance. Foram testadas três instâncias diferentes de escolas hipotéticas com cinco, seis e oito turmas. Para cada instância, testou-se os parâmetros com taxa 25%, 50% e 75%, tanto de crossover como de mutação.

Para garantir certa rigidez estatística, todos os testes foram repetidos dez vezes sob as mesmas condições e uma média aritmética simples foi calculada para avaliar a performance. A tabela 1 ilustra a média das gerações necessárias para se obter uma solução sob diferentes configurações de parâmetros.

Observou-se o melhor desempenho para o algoritmo com parâmetros de cruzamento em 50% e de mutação em 75% (menor média necessária e menor desvio-padrão geral).

7. Testes de Elitismo

A técnica do elitismo consiste em identificar o melhor indivíduo em uma população em um dado momento e automaticamente selecioná-lo para compor a próxima geração (seja com uma ou mais cópias). Na versão base de nosso algoritmo foi implementado o elitismo

com oito cópias do melhor indivíduo e, nesta seção, espera-se obter empiricamente qual seria o número ótimo de cópias para uma melhor performance.

A tabela 2 ilustra os testes realizados com o elitismo atribuído a 8, 10 e 15 cópias do melhor indivíduo, respectivamente. Os testes foram repetidos dez vezes para maior rigidez estatística.

Execução	8 indivíduos	10 indivíduos	15 indivíduos
1	1481	1417	4825
2	605	990	622
3	1376	569	581
4	1766	1150	1290
5	2665	1055	3253
6	1119	1733	924
7	919	838	1219
8	2408	1030	1955
9	999	3595	1948
10	1807	792	2216
Média	1514	1316	1883
Desvio	657	864	1318

Table 2. Desempenho do elitismo sob variados parâmetros. Dados expressos em número de gerações.

Nota-se que o melhor resultado foi obtido com 10 cópias do melhor indivíduo, um parâmetro intermediário entre o preenchimento da população com bons indivíduos e boa variabilidade genética provinda de indivíduos com score mais baixo.

8. Testes do Algoritmo Híbrido

Em um estudo datado de 2020, um grupo de pesquisadores [Katoch et al. 2020] realizou um levantamento (survey) sobre algoritmos genéticos, onde foi apontado o bom desempenho de uma técnica híbrida em AG's quando aplicada a problemas de agendamentos em geral. Tal técnica consiste em utilizar-se do algoritmo orinal para chegar a um ponto de quase solução (quando a população atinge um fitness médio pré-estabelecido) e então obter um resultado aplicando métodos convencionais de busca (local search).

Foi implementada uma versão de tal algoritmo híbrido para testes de performance executados sobre os parâmetros obtidos nas seções anteriores e com o elitismo de dez indivíduos. Nesta versão, utiliza-se o algoritmo genético padrão para obter uma solução contendo até dois conflitos de horários para então chamar o método de busca convencional.

Foi observado, porém, um problema inerente no que diz respeito ao comportamento do método de busca local e a forma com que ele interage com a instância de grade horária deste trabalho. A simples troca de posições na correção de um conflito, na maioria dos casos, acaba gerando outros conflitos em outras colunas da tabela. Isso acarretou em uma execução infinita da aplicação em todos os testes feitos, nunca convergindo a um resultado ideal.

9. Testes de Melhoria da População Inicial

Primeiramente foi implementado um método gerador com o auxílio de um contador, que verifica a quantidade de conflitos existentes em uma mesma coluna da grade antes de alocar uma turma. Caso hajam mais do que três conflitos, o algoritmo tenta gerar outro número aleatório, até um máximo de 10 tentativas. Caso nenhuma tentativa seja bem-sucedida, é alocado um número aleatório (respeitando a quantidade de aulas que cada professor precisa lecionar), mesmo se isso acarretar em um conflito.

Verificou-se uma melhora considerável no score das populações iniciais. Os resultados de dez execuções do algoritmo com uma instância de 8 turmas são apresentados na tabela 3 (segunda coluna), onde observa-se um desempenho semelhante ao do algoritmo original, porém com um desvio padrão menos acentuado. Isso nos diz que, apesar de não possuir tanto impacto no número de gerações necessárias para se obter uma resposta, esta técnica viabiliza tempo de execução mais constante e menos sujeito à aleatoriedade do AG.

Visto que cada professor obrigatoriamente dará ao menos uma aula em cada turma, testou-se também uma variação de inicialização onde primeiramente popula-se as diagonais da matriz com números crescentes, com o intuito de alocar turmas de forma a garantir menos colisões. Com o preenchimento inicial realizado, a grade é então submetida ao processo comum de inicialização pelo método pseudo-aleatório padrão.

Observa-se pela tabela 3 (terceira coluna) que o método possui desempenho semelhante aos outros, com a especifidade de apresentar um desvio-padrão muito elevado. Isso significa que , da mesma forma que pode-se gerar uma resposta com pouquíssimas gerações, pode-se também levar um tempo muito maior para se alcançar a condição de parada.

Execução	Original	Contador	Diagonal
1	1417	1067	2268
2	990	978	859
3	569	1149	848
4	1150	903	2533
5	1055	2683	1072
6	1733	1729	659
7	838	1361	1966
8	1030	665	1292
9	3595	1502	1790
10	792	1211	315
Média	1316	1324	1360
Desvio	864	657	1318

Table 3. Desempenho das duas técnicas empregadas em comparação ao modelo original. Resultados expressos em número de gerações. Testes executados com elitismo de 10 indivíduos e uma instância de 8 turmas.

10. Conclusões

O algoritmo genético cumpriu seu objetivo principal de resolver um problema combinatorial de alta complexidade em um tempo razoável de execução. Uma solução que satisfaz as restrições rígidas foi produzida em todos os testes. Tal solução, ainda que não seja a ideal, seria satisfatória para um colégio de porte pequeno.

Verificamos experimentalmente que a geração de uma população inicial melhor não necessariamente acarreta em um menor tempo de execução para o programa. A complexidade parece estar mais atrelada ao final da execução do algoritmo, onde instâncias quase solucionadas estão presentes na população, e a chance de resolver-se os conflitos restantes é muito baixa.

O estudo do elitismo mostrou que um balanço entre dominância genética por parte do indivíduo e variabilidade é o ideal para se alcançar uma boa solução no menor número de gerações possível. O fator de dez indivíduos estarem presentes na próxima geração, com nove sendo aptos para crossover e mutação foi importante para melhorar o score de gerações futuras e ao mesmo tempo garantir a permanência do indivíduo dominante na população.

Testes extensivos possibilitaram a obtenção de parâmetros ideais para as taxas de mutação e crossover, a fim de otimizar o número de gerações necessárias para resolver-se o problema. Pelos resultados, conclui-se que, para a instância do problema em questão, a variabilidade genética deve ser bastante favorecida, ainda mais tendo em vista que o problema de máximos locais é inexistente nesta representação.

O teste de um algoritmo híbrido falhou ao prover uma solução adequada ao problema em um tempo não-proibitivo. Especificidades da implementação fizeram com que o algoritmo de busca local entrasse em execução infinita e nunca retornasse uma resposta. Cabe, porém, uma exploração mais aprofundada em torno desta temática. Novas tentativas de otimizar o algoritmo de busca de forma a driblar os problemas encontrados podem ser promissoras.

References

Burke, E., Elliman, D., and Weare, R. (1994). A genetic algorithm for university timetabling. *AISB Workshop on Evolutionary Computation*, page Workshop Notes.

Corlone, A., Dorigo, M., and Maniezzo, V. (1990). Genetic algorithms and highly constrained problems, the timetabling case. *Parallel Problem Solving From Nature*, 6:55–59.

Fang, H.-L. (1994). Genetic algorithms in timetabling and scheduling.

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman Co., USA.

Katoch, S., Singh Chauhan, S., and Kumar, V. (2020). A review on genetic algorithm: past, present, and future.

Melanie, M. (1999). *An Introduction to Genetic Algorithms*. Massachusetts Institute of Technology, Cambridge, 5 edition.

Paechter, B., Luchian, H., Cumming, A., and Petruic, M. (1994). Two solutions to the general timetabling problem using evolutionary methods. *Proceeding of the First IEEE Conference on Evolutionary Computation*, pages 300–305.

Rezaeipanah, A., Matoor, S. S., and Ahmadi, G. (2020). A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search. *Applied Intelligence*, 51:467–492.

Ullman, J. D. (1973). Polynomial complete scheduling problems. *ACM SIGOPS Operating Systems Review*, 7(4):96–101.