Algoritmo genético para cálculo da tabela de horários dos Componentes Curriculares do curso de Ciência da Computação da UFFS

Gabriel Batista Galli[1], **Matheus Antonio Venancio Dall'Rosa**[1], **Vladimir Belinski**[1]

[1]Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS)
Caixa Postal 181 – 89.802-112 – Chapecó – SC – Brasil

{g7.galli96, matheusdallrosa94, vlbelinski}@gmail.com

Resumo. *O presente trabalho, apresentado ao curso de Ciência da Computação da Universidade Federal da Fronteira Sul - UFFS - Campus Chapecó - como requisito parcial para aprovação no Componente Curricular Inteligência Artificial, 2017.1, sob orientação do professor José Carlos Bins Filho, consiste em uma descrição detalhada da implementação de um algoritmo genético para calcular a tabela de horários dos Componentes Curriculares do curso de Ciência da Computação da UFFS.*

1. Descrição geral do algoritmo

A implementação do algoritmo genético encontra-se distribuída em dois arquivos: *main.cpp*, o arquivo principal do trabalho, cujo conteúdo engloba as funções do programa; e *ia.h*, o arquivo de cabeçalho de *main.cpp*, onde se encontram declaradas as estruturas utilizadas e alguns protótipos de funções.

O fluxo de execução do algoritmo pode ser encontrado na função *main()* de *main.cpp*. Inicialmente, realiza-se uma chamada à função *read_professors()*, que, a partir do arquivo de entrada, faz a leitura dos professores e dos horários que esses não desejam ministrar aulas.

Em seguida, é calculada a *fitness* máxima de um indivíduo, utilizada na função *fitness()* e em algumas verificações ao longo do código. Seu cálculo é dado da seguinte maneira: inicia-se com uma *fitness* de 450 pontos, dos quais 30 consistem em pontos atribuídos ao professor pelo fato dele não estar dando aula nos horários que não deseja, 30 pelo fato de não estar dando aulas nos períodos matutino e noturno no mesmo dia, 30 por não estar dando aulas sequenciais e 360 por não estar dando aula no último horário de uma noite e primeiro do dia seguinte. Os três primeiros casos se relacionam às preferências estabelecidas na descrição do trabalho proposto e o último corresponde à última restrição elencada. Cabe destacar que, devido a maneira em que o algoritmo foi implementado, as demais restrições não são violadas.

É importante enfatizar que a geração de um indivíduo que viole a restrição relacionada a um professor dar aula no último horário de uma noite e no primeiro horário do dia seguinte foi permitida, pois dado o modo como a população inicial é gerada, seu tratamento implicaria em perdas na eficiência do algoritmo. Contudo, através da realização dos testes foi verificado que a violação dessa restrição raramente ocorre na geração de um indivíduo, sendo que nenhum indivíduo apresentado como resposta final apresentou violações a essa restrição. Além disso, dada a forma como são selecionados indivíduos

para o cruzamento, um indivíduo que viole uma restrição apresentará uma probabilidade de cruzamento extremamente baixa. Logo, apesar de poder acontecer, a violação dessa restrição pouco ocorre e não impacta negativamente o resultado final do algoritmo.

Ao ser realizada uma violação da preferência que nos informa que um professor não deveria dar aula nos horários que não deseja, é realizado um desconto de 1 ponto por horário infringido (dentro de um limite de 30 possíveis horários de uma semana, motivo pelo qual a *fitness* do indivíduo inicia com 30 pontos atrelados a essa preferência). Por sua vez, ao ser violada a preferência que indica que um professor não deveria dar aula nos turnos matutino e noturno no mesmo dia, é realizado um desconto de 6 pontos na *fitness* do indivíduo por dia infringido (são 6 pontos, pois temos inicialmente 30 pontos para essa preferência e cinco dias na semana. Logo, $30/5 = 6$). Adicionalmente, ao ser realizada uma violação da preferência que apresenta que um professor não deveria dar aulas de maneira sequencial (um turno inteiro), é realizado um desconto de 2 pontos por turno infringido (são 2 pontos, pois temos inicialmente 30 pontos para essa preferência e 15 possíveis casos de aulas sequenciais por semana. Logo, $30/15 = 2$). Às três preferências foram atribuídos 30 pontos iniciais para que, assim, elas possuíssem o mesmo peso no cálculo da *fitness*.

Por fim, a restrição que informa que um professor não pode dar aula no último horário de uma noite e no primeiro horário da manhã seguinte é levada em conta na função de *fitness* da seguinte forma: inicialmente são atribuídos 360 pontos na *fitness* de um indivíduo para a não violação dessa restrição. A cada infração dela, 90 pontos são descontados da *fitness* do indivíduo (são 90 pontos, pois temos 360 pontos iniciais e 4 possibilidades de violação – $(21, 02)$, $(23, 04)$, $(25, 06)$, $(27, 08)$. Logo $360/4 = 90$). São 360 pontos para a restrição, pois, como uma violação de restrição não deve ocorrer, foi decidido que 80% da *fitness* seria atribuída a ela, de maneira que indivíduos que tivessem a restrição violada (caso com poucas ocorrências nos testes realizados) sofressem significativas penalidades e consequentemente tivessem sua probabilidade de reprodução amplamente diminuída.

Devido ao fato de não terem sido utilizados números de ponto flutuante para a representação da *fitness* e também para fins de normalização de seu cálculo, o valor inicial máximo da *fitness* (explanado acima) é multiplicado pelo número de professores dado na entrada.

Posteriormente ao cálculo da *fitness* máxima de um indivíduo, é calculada a *fitness* máxima da população, utilizada na função *fitness()* e em algumas impressões no console que visam auxiliar no acompanhamento da execução do código. O valor da *fitness* máxima da população é dado pela multiplicação da *fitness* máxima de um indivíduo pelo tamanho da população.

Cabe enfatizar que as funções relacionadas ao cálculo da *fitness* de um indivíduo, de uma população e de suas penalidades são, respectivamente, *fitness()*, *set_population_fitness()*, *count_schedules_to_avoid_infringements()*, *count_morning_night_infringements()*, *count_consecutive_schedules_infringements()* e *count_restriction_infringements()*, todas presentes em *main.cpp*.

Em sequência, são lidos do arquivo de entrada os cursos/semestres, salas e horários disponíveis para as salas (através da função *read_room_schedules()*), bem como

os CCRs, períodos, cursos/semestres e professores (através da função *read_subjects()*).

Após isso, é realizada a geração da população inicial, armazenada na estrutura *population_t* e montada através da chamada das funções *generate_people_permutation()* e *generate_population()*.

Na função *generate_people_permutation()* é realizada a construção de m permutações aleatórias, sem uso de *backtracking*. Seja $Professors = \{t_1, t_2, ..., t_m\}$ o conjunto de professores recebidos na entrada e uma permutação P_i definida por $P_i = \{p_{i_1}, p_{i_2}, ..., p_{i_m}\}$, é feito $p_{i_1} = t_i$, ou seja, associa-se o professor t_i ao primeiro elemento da permutação p_i, e sorteiam-se os $m - 1$ professores restantes sem uso de *backtracking* para as posições $\{p_{i_2}, ..., p_{i_m}\}$ da permutação P_i.

A partir disso, na função *generate_population()*, que faz uso das das permutações geradas em *generate_people_permutation()*, a construção dos indivíduos (com tamanho igual ao parâmetro constante POPULATION_SIZE) acontece da seguinte forma: a população é dividida em m grupos de tamanho $g =$ POPULATION_SIZE$/m$. Um grupo m_i é construído a partir da permutação i, ou seja, a primeira permutação de m_i é $m_{i_1} = P_1$. Então, são geradas $g - 1$ permutações lexicograficamente ordenadas a partir da permutação P_i, sendo utilizado para tal o algoritmo implementado na função *next_permutation()* da biblioteca *algorithm*. Assim, são gerados $m * g$ indivíduos, ou seja, POPULATION_SIZE indivíduos, cada um construído a partir de uma permutação.

Por sua vez, a atribuição de salas e horários para cada professor é feita através da ordem do conjunto $schedules$. Seja $schedules$ um conjunto de pares de sala e horário e $schedule_i$ e $schedule_j$ elementos deste conjunto, o conjunto $schedules$ é ordenado de tal forma que $schedule_i < schedule_j$ e $i < j$, caso a sala de $schedule_i$ possua menos horários disponíveis do que a sala de $schedules_j$. Dada a ordenação apresentada, à cada indivíduo é realizada a atribuição de seus horários, sendo esses tomados de maneira sequencial a partir de $schedule_1$, isto é, para cada permutação de professores são atribuídos os pares de sala e horário do conjunto $schedules$.

A seguir será provido um exemplo de execução das funções *generate_people_permutation()* e *generate_population()* utilizando a notação apresentada acima.

Em *generate_people_permutation()* seria realizado o que segue. Suponha uma entrada com 5 professores: Andressa, Bins, Claunir, Doglas e Emilio. Pela notação anterior teríamos então $m = 5$ e $Professors = \{Andressa, Bins, Claunir, Doglas, Emilio\}$. Ademais, como na definição das permutações temos que $p_{i_1} = t_i$, com certeza temos $P_1 = \{Andressa, -, -, -, -\}$, $P_2 = \{Bins, -, -, -, -\}$, $P_3 = \{Claunir, -, -, -, -\}$, $P_4 = \{Doglas, -, -, -, -\}$ e $P_5 = \{Emilio, -, -, -, -\}$, onde as posições marcadas com '$-$' são preenchidas com as outras possibilidades de professores de maneira aleatória.

Suponha, então, que o preenchimento aleatório tenha resultado em:

$$P_1 = \{Andressa, Bins, Claunir, Doglas, Emilio\}$$
$$P_2 = \{Bins, Andressa, Claunir, Doglas, Emilio\}$$
$$P_3 = \{Claunir, Andressa, Bins, Doglas, Emilio\}$$
$$P_4 = \{Doglas, Andressa, Bins, Claunir, Emilio\}$$
$$P_5 = \{Emilio, Andressa, Bins, Claunir, Doglas\}$$

Tendo sido geradas as permutações em *generate_people_permutation()*, na função *generate_population()* são gerados os indivíduos que comporão a população fazendo uso das permutações anteriormente criadas. Assim, supondo que o tamanho da população seja igual a 10 indivíduos (POPULATION_SIZE $= 10$) e sendo $m = 5$ (como já utilizado na função anterior), são criados 5 grupos compostos cada um por 2 indivíduos (POPULATION_SIZE$/m = 10/5 = 2$). Como um grupo m_i é construído a partir da permutação i, tem-se com certeza os grupos $m_1 = \{P_1, -\}$, $m_2 = \{P_2, -\}$, $m_3 = \{P_3, -\}$, $m_4 = \{P_4, -\}$, $m_5 = \{P_5, -\}$, onde '$-$' é uma permutação gerada pela ordenação lexicográfica da permutação fixa (P_i) de seu conjunto. Para a geração dessa permutação faz-se uso da função *next_permutation()* da biblioteca *algorithm*. Para os exemplos acima teríamos como grupos:

$$m_1 = \{\{Andressa, Bins, Claunir, Doglas, Emilio\},$$
$$\{Andressa, Bins, Claunir, Emilio, Doglas\}\}$$
$$m_2 = \{\{Bins, Andressa, Claunir, Doglas, Emilio\},$$
$$\{Bins, Andressa, Claunir, Emilio, Doglas\}\}$$
$$m_3 = \{\{Claunir, Andressa, Bins, Doglas, Emilio\},$$
$$\{Claunir, Andressa, Bins, Emilio, Doglas\}\}$$
$$m_4 = \{\{Doglas, Andressa, Bins, Claunir, Emilio\},$$
$$\{Doglas, Andressa, Bins, Emilio, Claunir\}\}$$
$$m_5 = \{\{Emilio, Andressa, Bins, Claunir, Doglas\},$$
$$\{Emilio, Andressa, Bins, Doglas, Claunir\}\}$$

Agora, supondo que existam as salas 101 (com os horários $00, 02, 03, 05, 15, 21$ disponíveis), 102 (com os horários $00, 05, 11, 23$ disponíveis) e 103 (com os horários $05, 07, 15, 16, 17$ disponíveis), é realizada a ordenação de seus respectivos pares de sala e horário de maneira não decrescente pela quantidade de horários disponíveis em cada sala. Dessa forma, se $schedules$ é o conjunto de pares de sala e horário, como resultado da ordenação teríamos $schedules = \{pares\_102, pares\_103, pares\_101\}$, onde $pares\_101$, $pares\_102$ e $pares\_103$ correspondem aos pares de sala e horário para as salas 101, 102 e 103, respectivamente. Feito isso, para cada uma das permutações de professores (m_1 à m_5) são então associadas de maneira sequencial os pares de sala e horários aos professores, obviamente sendo cuidado para não ocorrerem violações de restrições e serem feitas associações válidas.

Dessa forma, é feita a geração de toda uma população inicial diversificada.

Finalmente, é realizada uma chamada à função *evolve()*, que é é responsável pela evolução da população e recebe como parâmetro a quantidade de gerações em que o algoritmo será executado. A cada geração uma nova população é inicializada, sendo os indivíduos dessa nova população resultantes do cruzamento de dois indivíduos da população atual, que, por sua vez, são sorteados de maneira com que os indivíduos que apresentem maior *fitness* tenham maior probabilidade de reprodução. São realizados cruzamentos até que a nova população atinja o tamanho da atual. Após cada cruzamento (detalhado na seção "Cruzamento dos indivíduos") é realizada a mutação do indivíduo resultante dentro de uma probabilidade especificada no código (a mutação será detalhada na seção "Mutação dos indivíduos").

Ainda a respeito da implementação do algoritmo, cabe destacar os critérios de parada estabelecidos para esse. O algoritmo é finalizado quando ocorre um dos seguintes casos:

1. Quando é gerado um indivíduo de *fitness* máxima, ou seja, uma resposta ideal/perfeita ao problema, na qual não haveria nenhuma violação de restrições e preferências;
2. Quando a *fitness* de uma população se mantém inalterada por um número estabelecido de gerações, definido no parâmetro constante MAX_UNMODIFIED_GENERATIONS;
3. Quando o algoritmo atinge um número máximo de gerações, definido no parâmetro constante MAX_GENERATIONS.

Cabe destacar que ao longo da execução do algoritmo são impressas no console a numeração das gerações, a *fitness* de sua população e a *fitness* máxima. Ao ser atingido um dos critérios de parada informados, também é destacado por qual motivo a execução foi finalizada e impressos no console os dados do indivíduo de maior *fitness* da última população gerada. Por fim, cabe destacar que para fins de depuração pode ser descomentado o define de depuração (#DEFINE DEBUG) no arquivo *main.cpp*, o qual faz com que sejam impressas mais informações no console durante a execução do algoritmo.

2. Representação de um indivíduo

Um indivíduo é representado pela estrutura *person_t*, presente em *ia.h*, sendo definido por uma lista de genes e um valor de *fitness*.

Por sua vez, cada gene é representado pela estrutura *schedule_t*, presente em *ia.h*, que armazena as informações referentes à sala de aula (estrutura *room_t*, que contém o número da sala, o curso e os horários disponíveis da sala), ao componente curricular (estrutura *subject_t*, onde também pode ser encontrado, por exemplo, o professor associado ao componente curricular e o curso), ao horário da semana e o número do período.

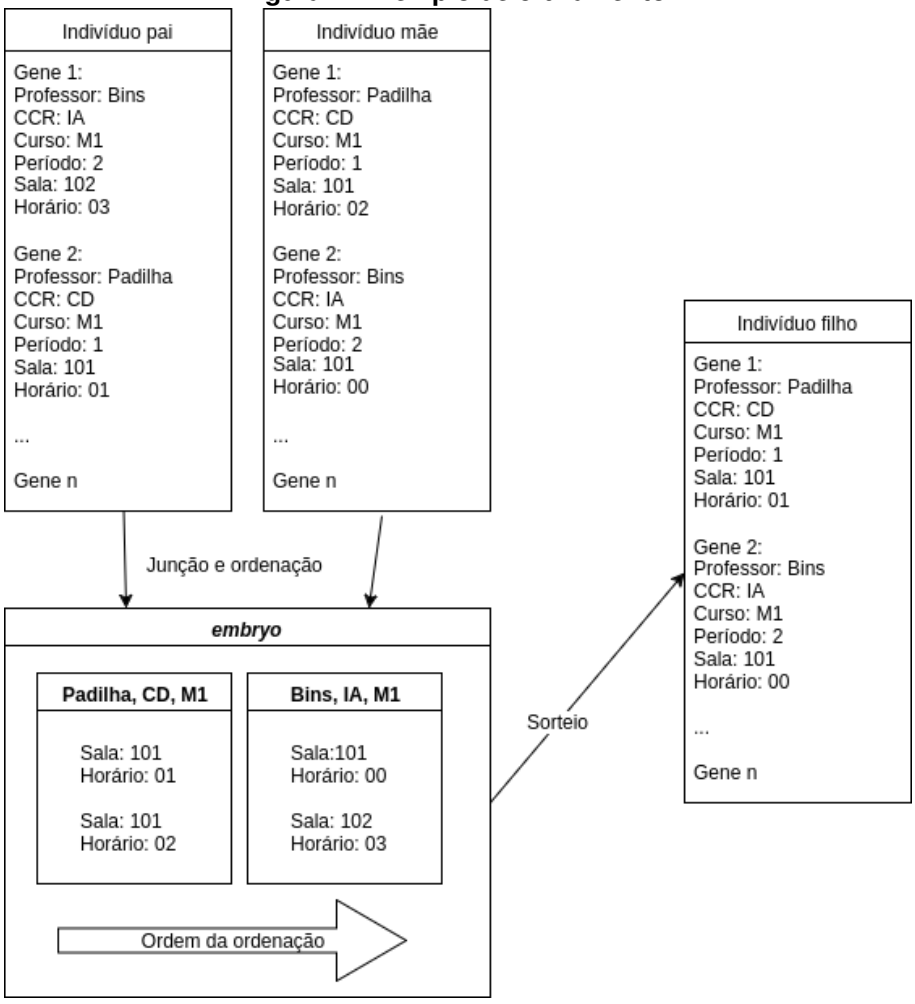
3. Cruzamento dos indivíduos

A implementação do cruzamento pode ser encontrada na função *cross()* de *main.cpp*. A chamada de *cross()* é realizada dentro de *evolve()*, de forma a gerar os indivíduos de uma nova população. *cross()* recebe como parâmetros um indivíduo pai e um indivíduo mãe, sorteados da forma apresentada na seção "Descrição geral do algoritmo".

Inicialmente, são agrupados em uma estrutura nomeada *embryo* os pares sala e horário do indivíduo pai e do indivíduo mãe que apresentam a mesma tripla professor, componente curricular e curso. Após isso, *embryo* é ordenado de forma não decrescente pela quantidade de períodos de um determinado componente curricular. O desempate da ordenação é realizado pela quantidade de salas disponíveis, de forma com que as triplas com maior quantidade de salas são ocupadas primeiro.

Em seguida, para cada tripla professor, componente curricular e curso de *embryo*, são sorteados *n* pares de sala e horário, sendo *n* a quantidade de períodos necessários ao componente curricular da tripla. Após cada sorteio, são realizadas então verificações de forma a garantir que um professor não ministre duas aulas no mesmo horário ou ocupe uma sala já reservada à outra tripla professor, CCR e curso. Assim, ao final do cruzamento é tido como resultado um indivíduo válido e que apresenta em seu cromossomo uma mistura de genes de seus indivíduos pai e mãe.

Figura 1. Exemplo de cruzamento.



Um exemplo simplório de cruzamento é apresentado na Figura 1 (cabe destacar que na imagem são apresentadas somente as informações necessárias para a explicação). Nele, temos um indivíduo pai e um indivíduo mãe, cada um com uma sequência de genes. Conforme apresentado anteriormente, o passo inicial da função de cruzamento é reali-

zar o agrupamento na estrutura *embryo* dos pares sala e horário do indivíduo pai e do indivíduo mãe que apresentam a mesma tripla professor, componente curricular e curso. No exemplo da Figura 1 percebe-se que para a tripla Padilha, CD, M1 têm-se os pares sala e horário $(101, 01)$ e $(101, 02)$ e para a tripla Bins, IA, M1 têm-se os pares $(101, 00)$ e $(102, 03)$. Para o exemplo, a ordenação apresenta o CCR CD antes do CCR IA, pois, conforme apresentado como critério de ordenação, CD apresenta um número menor de períodos do que IA. Após a montagem de *embryo* são realizados os sorteios para tomada dos genes que irão compor o cromossomo do indivíduo filho. No exemplo é apresentado no indivíduo filho o resultado de um possível sorteio, no qual temos que o esse herdou genes do pai e da mãe.

4. Mutação dos indivíduos

A implementação da mutação pode ser encontrada em *mutate()* de *main.cpp*. Tal função é executada após o cruzamento de dois indivíduos e ocorre seguindo a probabilidade de mutação estabelecida no parâmetro constante MUTATION_RATE.

A função *mutate()* recebe um indivíduo candidato à sofrer uma mutação e constrói uma estrutura que armazena os horários atualmente livres dentre as salas de aula dadas na entrada do problema. Então, itera pelos horários alocados no indivíduo sendo analisado e, ao ser encontrada uma sala de aula com horário disponível e de mesmo curso (e.g. V1...) que o horário sendo considerado no momento, sorteia um horário dentre os disponíveis e realiza a troca de sala e horário no cromossomo do indivíduo.

5. Parâmetros utilizados na execução do algoritmo

No total, quatro parâmetros constantes (definido em *main.cpp*) são utilizados ao longo do algoritmo, sendo eles:

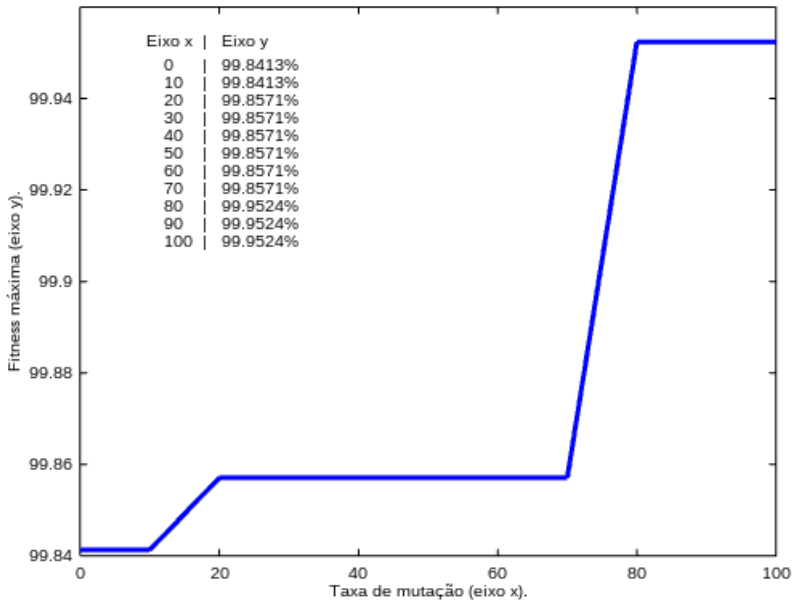
1. POPULATION_SIZE: representa o tamanho da população;
2. MUTATION_RATE: representa a taxa de mutação, a qual varia de 0 a 100;
3. MAX_GENERATIONS: representa o número máximo de gerações pelas quais o algoritmo será executado, caso os demais critérios de parada não sejam satisfeitos antes;
4. MAX_UNMODIFIED_GENERATIONS: representa o número máximo de gerações sem alteração de *fitness*, considerado como um dos critérios de parada do algoritmo.

6. Resultado final

Nesta seção serão descritos alguns dos testes efetuados para a validação do presente trabalho, bem como seus respectivos resultados. As entradas mencionadas podem ser verificadas em '/src/input'.

Um dos testes realizados consistiu na execução do algoritmo para a entrada *1.in* a fim de verificar a relação entre a taxa de mutação e a *fitness* dos indivíduos. Para isso, o algoritmo desenvolvido foi executado com a entrada *1.in* para um tamanho de população igual a 80 indivíduos e para valores de taxa de mutação iguais a 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% e 100%. Os resultados obtidos podem ser verificados na Figura 2.

Figura 2. Taxa de mutação (%) X *Fitness* máxima



A partir da análise da Figura 2 pode-se notar que para o problema em questão e a implementação realizada, quanto maior a taxa de mutação, maior a *fitness* máxima encontrada em uma população. Cabe destacar que, em relação aos valores do eixo y do gráfico, 100% corresponde ao valor máximo de *fitness* que poderia ser atingido por um indivíduo.

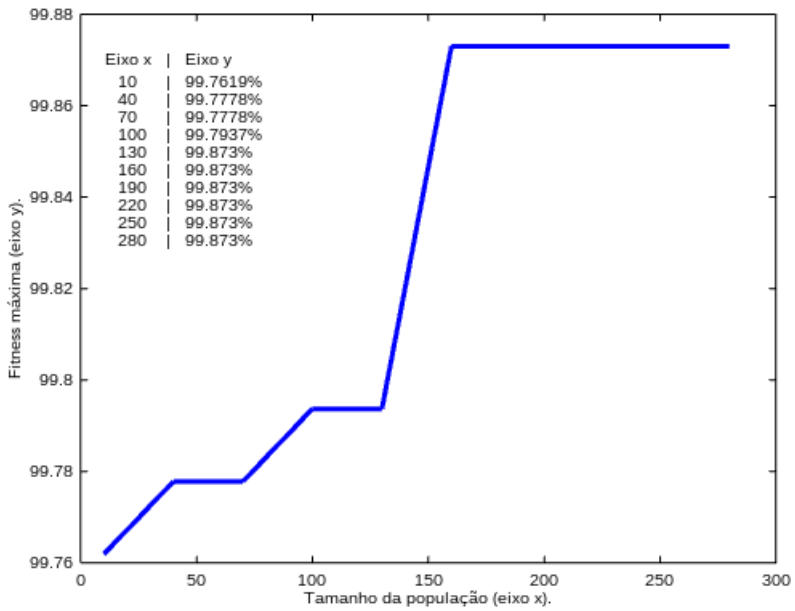
Outro teste realizado consistiu na execução do algoritmo para a entrada *1.in* a fim de verificar a relação entre o tamanho da população e a *fitness* dos indivíduos. Para isso, o algoritmo foi executado para a entrada *1.in* com uma taxa de mutação igual a 90% e para valores de tamanhos de população iguais a 10, 40, 70, 100, 130, 160, 190, 220, 250 e 280 indivíduos. Os resultados obtidos podem ser verificados na Figura 3.

A partir de análise da Figura 3, pode-se notar que para o problema em questão e a implementação realizada, quanto maior o tamanho da população, maior é a *fitness* máxima encontrada em uma população. Cabe destacar que, em relação aos valores do eixo y do gráfico, 100% corresponde ao valor máximo de *fitness* que poderia ser atingido por um indivíduo.

A seguir serão comentados exemplos de saídas para as entradas presentes em *1.in* e *2.in*. Cabe destacar que as saídas dos testes a seguir foram obtidas por meio das informações que estão sendo impressas no console por padrão: tamanho da população, quantidade máxima de gerações, *fitness* máxima da geração inicial, *fitness* máxima da última geração, indivíduo de maior *fitness* da última geração (com sua *fitness*, horários, quantidade de preferências e restrições violadas – mesmo nunca tendo sido violada uma restrição no melhor indivíduo).

A entrada *1.in* é formada pela declaração de 14 professores, 9 salas e 27 associações de professores e componentes curriculares. Ademais, para essa entrada o número total de períodos é igual a 44, o número de total de horários disponíveis nas salas

Figura 3. Tamanho da população (em indivíduos) X *Fitness* máxima



é igual a 70, o número de preferências de professores em não dar aula em determinado horário é igual a 61 e o número total de preferências a serem considerados pelo programa é igual a 81.

Para uma execução do algoritmo genético implementado tomando *1.in* como entrada, uma taxa de mutação de 90%, um tamanho de população igual a 160 indivíduos e 100 gerações (valores percebidos através dos testes como geradores de bons resultados, o que inclusive pode ser verificado nas figuras anteriormente exibidas, e por isso escolhidos), obteve-se na primeira geração como indivíduo com melhor *fitness* um representante com *fitness* igual a 99.76% da máxima possível (ou seja, caso em que não ocorreria nenhuma violação de preferências). Por sua vez, obteve-se na última geração como indivíduo com melhor *fitness* um representante com *fitness* igual a 99.87% da máxima possível.

Esse indivíduo satisfazia todas as restrições estabelecidas e violava apenas 3 preferências, duas relacionadas a um professor ter sido alocado em um horário para o qual desejava não dar aula (no teste, o professor Emílio para o horário 01 e o professor Bins para o horário 00) e uma relacionada a um professor dar aula nos turnos matutino e noturno no mesmo dia (no teste, a professora Laís para os horários 04 e 24). Os dados completos desse indivíduo podem ser visualizados na Tabela 1. Cabe destacar que na Tabela 1 o valor da coluna "Período" é dado da seguinte maneira: caso na entrada tenha sido informado que um professor irá dar 1 período de um componente curricular, na saída o valor 0 indica que com aquela alocação restam 0 períodos para se atender a demanda da entrada; se na entrada foi informado que um professor dará 2 períodos de um componente curricular, na saída os valores 1 e 0 indicam que com aquela alocação em questão restam 1 e 0 períodos para se atender a demanda da entrada – note que nesses casos sempre há uma alocação para 1 e uma para 0, ou seja, todos os períodos solicitados são alocados; para quantidades maiores de períodos as considerações são análogas às anteriores.

Tabela 1: Exemplo de saída para a entrada de *1.in*

Período	**Código do CCR**	**Professor(a)**	**Curso/semestre**	**Sala**	**Horário**
0	ES1	Raquel	M5	103	5
1	ES1	Raquel	M5	103	7
0	BD2	Denio	M5	103	3
1	BD2	Denio	M5	103	8
0	SD	Emilio	N2	102	27
1	SD	Emilio	N2	102	25
0	*ED2*	*Lais*	*M3*	*102*	*4*
1	ED2	Lais	M3	102	0
0	II	Raquel	V1	101	11
1	II	Raquel	V1	101	13
0	Alg	Priscila	V1	101	10
1	Alg	Priscila	V1	101	12
0	OC	Padilha	N4	104	27
1	OC	Padilha	N4	104	23
0	TCC1	Marco	M7	104	4
1	TCC1	Marco	M7	104	8
0	SO	Marco	M5	103	6
1	SO	Marco	M5	103	2
0	CD	Jacson	V1	101	16
1	CD	Jacson	V1	101	14
0	IPC	Graziela	M5	103	9
1	IPC	Graziela	M5	103	16
0	IA	Bins	M7	104	2
1	**IA**	**Bins**	**M7**	**104**	**0**
0	LFA	Braulio	M5	103	4
1	LFA	Braulio	M5	103	10
0	BD1	Denio	N4	104	21

(continua)

Tabela 1 – Exemplo de saída para a entrada de *1.in*

(continuação)

Período	**Código do CCR**	**Professor(a)**	**Curso/semestre**	**Sala**	**Horário**
1	BD1	Denio	N4	104	26
0	Dist	Emilio	M7	104	3
1	**Dist**	**Emilio**	**M7**	**104**	**1**
0	OC	Jacson	M3	102	1
1	OC	Jacson	M3	102	5
0	Opt1	Guilherme	M7	104	5
1	Opt1	Guilherme	M7	104	9
0	Prog2	Lais	N4	104	22
0	*ED1*	*Lais*	*N2*	*102*	*24*
0	Opt2	Padilha	M7	104	11
0	Prg1	Lais	M3	102	6
0	CD	Padilha	V1	101	15
0	Prg1	Priscila	M3	102	8
0	Prog2	Priscila	N4	104	20
0	Opt2	Andressa	M7	104	10
0	Alg	Guilherme	V1	101	18
0	ED1	Jacson	N2	102	26

O exemplo acima ilustra uma saída onde ocorreram algumas violações de preferências (em negrito para o caso de um professor ter sido alocado em um horário para o qual desejava não dar aula e em itálico – o par – nos casos em que um professor é alocado para lecionar nos turnos matutino e noturno no mesmo dia) e indica quais foram. Por sua vez, o exemplo a seguir demonstra a geração de um indivíduo para o qual foi obtida *fitness* máxima.

A entrada *2.in* é formada pela declaração de 11 professores, 6 salas e 14 associações de professores e componentes curriculares. Ademais, para essa entrada o número total de períodos é igual a 19, o número de total de horários disponíveis nas salas é igual a 52, o número de preferências de professores em não dar aula em determinado horário é igual a 44 e o número total de preferências a serem considerados pelo programa é igual a 64.

Para uma execução do algoritmo genético implementado tomando *2.in* como en-

trada, uma taxa de mutação de 90%, um tamanho de população igual a 160 indivíduos e 100 gerações (valores percebidos através dos testes como geradores de bons resultados, o que inclusive pode ser verificado nas figuras anteriormente exibidas, e por isso escolhidos), obteve-se um indivíduo com *fitness* igual a 100%, ou seja, que não violava nenhuma restrição e nenhum tipo de preferência. Esse indivíduo era composto pelos dados da Tabela 2, onde o valor da coluna "Período" é dado da mesma forma como explicado no exemplo anterior.

Tabela 2: Exemplo de saída para a entrada de *2.in*

Período	**Código do CCR**	**Professor(a)**	**Curso/semestre**	**Sala**	**Horário**
0	LF	Braulio	V1	101	11
1	LF	Braulio	V1	101	13
0	CD	Emilio	M2	202	7
1	CD	Emilio	M2	202	3
0	ES	Graziela	M1	201	6
1	ES	Graziela	M1	201	8
0	IC	Graziela	N2	201	22
1	IC	Graziela	N2	201	24
0	P2	Priscila	V1	101	14
1	P2	Priscila	V1	101	16
0	BD	Andressa	V1	101	10
0	CG	Bins	M1	201	0
0	IA	Bins	M1	201	2
0	P1	Doglas	M2	202	0
0	CD	Jacson	M1	201	4
0	OC	Jacson	N1	301	20
0	SO	Marco	V2	101	18
0	E1	Pavan	N2	201	20
0	GP	Raquel	V2	101	12

Por fim, cabe destacar que a partir dos testes realizados observou-se que o algoritmo implementado atendeu ao propósito ao qual foi desenvolvido, consistindo em um algoritmo genético para cálculo da tabela de horários dos Componentes Curriculares do curso de Ciência da Computação da UFFS.

Ademais, também foi observado que: (1) a população inicial gerada através das funções *generate_people_permutation()* e *generate_population()* já costuma apresentar alguns indivíduos com *fitness* alta para as entradas; (2) a *fitness* dos indivíduos aumenta ao passar das gerações; (3) a *fitness* dos indivíduos costuma convergir rapidamente; (4) para o algoritmo desenvolvido e as entradas do problema, taxas de mutação altas costumam apresentar resultados melhores; e (5) para o algoritmo desenvolvido e as entradas do problema, tamanhos de população maiores costumam apresentar resultados melhores.