Trabalho de Conclusão de Curso

Uma Abordagem de Algoritmo Genético para Agrupamento de Dados: Um Estudo de Caso sobre Dados Educacionais

Lucas Benevides Viana de Amorim
lucas@ic.ufal.br

Orientador:

Evandro de Barros Costa

Maceió, Fevereiro de 2011

Lucas Benevides Viana de Amorim

Uma Abordagem de Algoritmo Genético para Agrupamento de Dados: Um Estudo de Caso sobre Dados Educacionais

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientador:

Evandro de Barros Costa

Maceió, Fevereiro de 2011

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

Evandro de Barros Costa - Orientador
Instituto de Computação
Universidade Federal de Alagoas

Aydano Pamponet Machado - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Roberta Vilhena Vieira Lopes - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Lucas Benevides Viana de Amorim - Autor

Maceió, Fevereiro de 2011

Resumo

A tarefa de agrupamento consiste em determinar um conjunto finito de categorias (grupos) para descrever um certo conjunto de dados de acordo com a semelhança entre seus objetos. Por esta ser considerada uma tarefa de grande complexidade, tem-se procurado algoritmos que sejam eficientes para encontrar soluções próximas da otimalidade a um custo computacional aceitável.

Neste trabalho, apresentamos uma implementação de Algoritmos Genéticos para otimizar soluções do problema da formação de grupos de estudantes em uma turma de um curso na modalidade de ensino a distância. Certos aspectos relativos a este cenário, tais como a coesão sociométrica dos grupos gerados e a motivação dos estudantes, são considerados pela função de avaliação multi-objetivo composta. Um experimento é feito demonstrando a eficiência dessa abordagem.

Abstract

The task of clustering consists in determining a finite set of categories (groups) to describe a certain data set according to the similarities among its objects. Since this is considered a high complexity task, researchers have been looking for algorithms which are efficient in finding near-optimal solutions at an acceptable computational cost.

In this work, we present an implementation of Genetic Algorithms aimed at optimizing solutions to the problem of forming groups of students in a class of a distance learning course. Certain aspects related to this scenario, such as the sociometric cohesion and the students' motivations, have been considered by the composed multi-objective fitness function. An experiment have been performed in order to show the effectiveness of this approach.

Agradecimentos

Eu não poderia começar este texto sem ser agradecendo, primeiramente, aos meus pais, que detêm a maior parcela de culpa pelo que sou hoje, por todo amor e dedicação e pelo bem de valor inimaginável que proporcionaram durante a minha vida, a educação. Agradeço-lhes também por terem me apoiado de todas as formas que lhes eram alcançáveis durante os anos de minha graduação, me incentivando e motivando nos momentos de dúvida. Agradeço imensamente a toda minha família, meus irmãos, primos, tios e avós, os de perto e os de longe, que sempre me trouxeram alegria e incentivo.

Agradeço a todos os professores que estiveram presentes em algum ponto da trajetória de minha graduação, ao professor Jaime Evaristo, através de quem tive o meu primeiro contato com a computação quando em suas aulas de Programação 1 e quem até hoje me inspira por sua paixão à profissão. Ao professor Patrick Henrique, um exemplo de didática, profissionalismo e bom humor com quem me aconselhei em diversas decisões que precisei tomar. Agradeço ao meu orientador, professor Evandro de Barros por seu tempo e esforço dedicados em me apresentar boa parte do que sei sobre Inteligência Artificial e Mineração de dados, e por ter me oferecido um tema tão interessante de se trabalhar.

Devo agradecimentos também à professora Roberta Lopes, quem teve o prazer de me introduzir aos seus queridos Algoritmos Genéticos e até hoje me ajuda a resolver os problemas mais difíceis. Ao professor Leandro Dias que me orientou enquanto bosista do COMPE, e ao professor Leandro Sales com quem tenho interessantíssimas discussões onde aprendo sobre redes de computadores, kernel Linux e outros temas não menos interessantes.

Aos meus colegas de turma, por toda a sua paciência em suportar minhas frequêntes perguntas durante as aulas do curso. Aos que me acompanharam mais de perto no laboratório, Camilo, Neto, Oswaldo, Sidney e Willian pelas agradáveis companhias e também por me ajudarem a construir meus conhecimentos de programação, e a Felipe ptcho e Vinícius que me ajudaram a resolver muitos dos bugs desse trabalho.

Um agradecimento todo especial a minha amada noiva, Ana Paula, a quem devo a maior parte do meu entusiasmo, alegria e força de vontade durante todo esse tempo em que me acompanha e principalmente nos momentos mais difícies da minha vida, compartilhando cada conquista e cada dificuldade, e a quem dedico este trabalho.

Lucas Amorim

Sumário

1 Introdução **1**

2 Fundamentação Teórica **3**
2.1 Algoritmos Genéticos 3
2.1.1 Funcionamento de um Algoritmo Genético 4
2.1.2 Representação da Solução 5
2.1.3 Etapas de um Algoritmo Genético 6
2.2 Sobre a Tarefa de Agrupamento de Dados 9

3 Abordagem Proposta **11**
3.1 O Problema 11
3.2 Representação das Soluções 12
3.3 Funcionamento do AGrupos 13
3.3.1 Geração da População Inicial 14
3.3.2 Função de Avaliação 14
3.3.3 Condição de Parada 16
3.3.4 Seleção 16
3.3.5 Operador de Cruzamento 16

4 Avaliação da Proposta **20**

5 Conclusão **27**

Referências **29**

Apêndices **30**

A Código do AGrupos **31**

B Base de Dados utilizada para avaliação do AGrupos **34**

Lista de Figuras

2.1 Fluxograma de um algoritmo genético 5
2.2 Representação vetorial de uma solução/cromossomo para o problema do caixeiro viajante, onde o vetor indica a sequência de nós a visitar. 6
2.3 Representação matricial de uma solução/cromossomo para o problema do caixeiro viajante onde a matriz indica a sequência de arestas a percorrer. 6
2.4 (a) Cruzamento de um ponto e (b) Cruzamento de dois pontos em um cromossomo de representação por meio de vetor binário. 8
2.5 Mutação em um cromossomo de representação por meio de vetor binário. . . . 8

3.1 Representação da solução como um conjunto de grupos de alunos. 12
3.2 Representação da população. 12
3.3 Fluxograma do AGrupos 13
3.4 Cruzamento Orientado a Grupos tal como implementado no AGrupos. 17

4.1 Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 0.01%. 21
4.2 Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 1%. 21
4.3 Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 4%. 22
4.4 Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 9%. 22
4.5 Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 15%. 23
4.6 Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 25%. 23
4.7 Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 30%. 24
4.8 Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 40%. 24
4.9 Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 50%. 25
4.10 Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 60%. 25
4.11 Variação da taxa de convergência da aptidão das soluções de acordo com a probabilidade de mutação. 26

Lista de Códigos

A.1 Cabeçalho da classe Estudante 31
A.2 Cabeçalho da classe Grupo 31
A.3 Cabeçalho da classe Solução 32
A.4 Função de Geração da População Inicial 32

Capítulo 1

Introdução

A tarefa de agrupamento de dados, também conhecida por particionamento, consiste em determinar um conjunto finito de categorias (grupos) para descrever um certo conjunto de dados de acordo com a similaridade entre seus objetos. Trata-se de uma tarefa requerida em vários domínios de aplicação, a exemplo de saúde e educação. Entretanto, a solução para tal tarefa apresenta-se com uma complexidade NP-Difícil [1], ou seja, pelo menos tão complexa quanto o mais complexo problema NP. Daí, tem-se procurado algoritmos que sejam eficientes para encontrar soluções próximas da otimalidade em tempo e esforço computacionais aceitáveis.

Por essa razão, os Algoritmos Genéticos (AG) têm sido muito utilizados para agrupamento [3, 4, 5, 6], e detêm nossa atenção justamente por serem reconhecidamente bem sucedidos no cenário onde é necessária uma rápida convergência para soluções próximas da otimalidade em grandes espaços de busca [3].

Neste trabalho propomos uma abordagem para AG, mostrando sua eficiência, para a tarefa de agrupamento de dados educacionais, tendo como foco o cenário de formação de grupos em uma turma de estudantes de ensino a distância. O cenário escolhido é adequado pois existe uma grande disponibilidade de dados provenientes do uso de Ambientes Virtuais de Aprendizagem (AVA), tais como Moodle [7] e Massayó [8].

Nesses ambientes, o estudante interage com o curso por meio de comentários em fóruns de discussão e resolução de questionários sobre os assuntos da disciplina, apresentados na forma de páginas da web ou objetos virtuais de aprendizagem. O estudante ainda é capaz de interagir com seus colegas, professores e tutores, através de mensagens privadas ou de comentários em seus blogs pessoais. Toda essa interação é registrada no banco de dados do AVA, que acaba se tornando uma importante fonte para mineração de dados relacionados a educação.

Esta monografia está dividida em quatro capítulos, sendo este o primeiro. No capítulo 2, apresentamos a parte de conhecimento de apoio a este trabalho. No capítulo 3, descrevemos a abordagem proposta onde detalharemos o cenário e aspectos relativos ao funcionamento

do algoritmo e à implementação do AGrupos, a nossa implementação do AG para tarefa de agrupamento no cenário descrito. E por fim, um experimento realizado com uma base de dados artificial a fim de avaliar a nossa proposta.

Capítulo 2

Fundamentação Teórica

Neste capítulo apresentamos uma síntese sobre Algoritmos Genéticos seguida de uma discussão sobre seu uso para a tarefa de agrupamento de dados.

2.1 Algoritmos Genéticos

Algoritmos Genéticos (AGs) são modelos computacionais de busca e otimização de soluções [1], inicialmente propostos por Holland [9], com forte inspiração na teoria da evolução das espécies de Charles Darwin [1]. Nos algoritmos genéticos, as soluções, que representam diferentes pontos no espaço de busca, são representadas por cromossomos artificiais dos quais os genes representam as características daquela solução. Uma série de analogias podem ser feitas entre os algoritmos genéticos e a evolução das espécies, conforme podemos ver na Tabela 2.1.

Evolução das Espécies	Algoritmos Genéticos
Meio Ambiente	Problema , Função de avaliação
Indivíduo	Solução
Cromossomo	Representação (string binária, vetor, lista)
Gene	Característica do Problema
Alelo	Valor da Caracterísitca
Reprodução Sexual	Operador de Cruzamento
Mutação	Operador de Mutação
População	Conjunto de Soluções
Gerações	Ciclos

Tabela 2.1: Analogia entre a evolução das espécies e os algoritmos genéticos [1].

Além da tarefa de agrupamento de dados, os algoritmos genéticos aplicam-se a uma miríade de tarefas, tais como a otimização numérica [10], otimização de distribuição e logísitca

[1] Na teoria da evolução das espécies os indivíduos mais aptos tem maiores chances de sobrevivência e portanto de gerar descendentes, perpetuando seu código genético.

[11, 12], seleção de atributos em dataminig [13, 14], sistemas de formação de células de fabricação [15, 16, 5] e outros.

2.1.1 Funcionamento de um Algoritmo Genético

O funcionamento do AG busca reproduzir um ambiente natural, onde somente os indivíduos mais aptos prosperam e reproduzem, transmitindo seu código genético para as próximas gerações, tal como descrito a seguir.

Inicialmente, um conjunto de soluções ou cromossomos, chamado de população, é criado de forma aleatória, constituindo-se então na população inicial que tem cada uma de suas soluções aferidas pela função de avaliação e associadas a um certo valor de aptidão. Baseado no princípio da seleção natural, as soluções mais aptas são selecionadas e submetidas aos operadores genéticos. Cada gene, ou característica, tem uma pequena probabilidade de sofrer mutação e cada solução uma outra probabilidade de sofrer cruzamento, o que poderá, ou não, melhorar a aptidão do indivíduo. Ao fim de cada ciclo, a aptidão dos indivíduos, ou seja, das soluções, é medida pela função de avaliação. Esse processo continua por um determinado número de ciclos ou até que a condição de parada seja satisfeita.

No Pseudo-código apresentado no Algoritmo 1 e no fluxograma da Figura 2.1, apresentamos a estrutura de um algoritmo genético, onde podemos notar que, basicamente, o algoritmo é composto de um laço principal (ciclo), que representa as gerações de indivíduos, no qual são executados os elementos básicos deste algoritmo: A função de avaliação, a seleção dos indivíduos a compor a nova população, os operadores genéticos (cruzamento e mutação), e a substituição da população antiga pela nova população gerada com indivíduos mais aptos. Mais à frente apresentaremos mais detalhes sobre cada uma dessas etapas.

Algoritmo 1 Algoritmo Genético

```
1: T = 0
2: Gerar população inicial P(0)
3: Avaliar P(0)
4: while Critério de parada não satisfeito do
5:     T = T + 1
6:     Selecionar P(T) a partir de P(T - 1)
7:     Aplicar operador de cruzamento
8:     Aplicar operador de mutação
9:     Avaliar P(T)
10: end while
```

Podemos notar que temos um processo paralelo e adaptativo de busca e otimização de soluções, uma vez que várias soluções são consideradas a cada iteração (já que uma população é um conjunto de soluções) e que essas tendem a evoluir a medida que se passam os

ciclos [2].

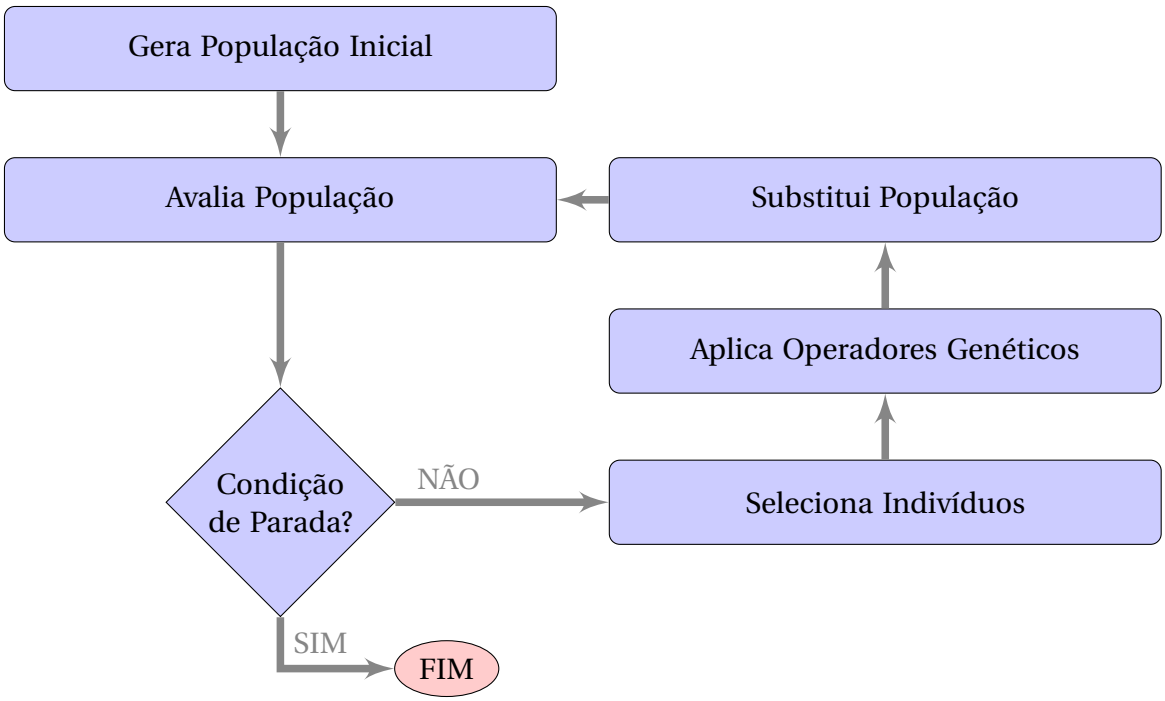

Figura 2.1: Fluxograma de um algoritmo genético

2.1.2 Representação da Solução

É possível representar a solução, ou seja, estruturar o cromossomo artificial, de diversas maneiras, desde vetores ou cadeias de números [3, 16] e matrizes de inteiros até tipos abstratos de dados [17, 18]. Segue-se uma breve descrição dessas representações.

Representação Vetorial

Esta é uma das representações mais simples, que pode ser usada para vetores de números inteiros ou binários e se aplica a uma larga gama de problemas. Como exemplo, demonstramos a representação de uma solução para o clássico problema do caixeiro viajante, conforme podemos ver na Figura 2.2

Representação Matricial

A depender do problema, a representação matricial pode ser mais intuitiva que a vetorial. No nosso exemplo do problema do caixeiro viajante poderíamos representar, em cada coluna da matriz, os extremos das arestas, de forma que, seguindo o caminho do mapa da Figura 2.2 teríamos a matriz da Figura 2.3 representando a sequência de arestas a serem percorridas.

[2] Esta evolução está diretamente ligada à convergência do algoritmo que por sua vez dependerá, principalmente, da função de avaliação e dos operadores genéticos.

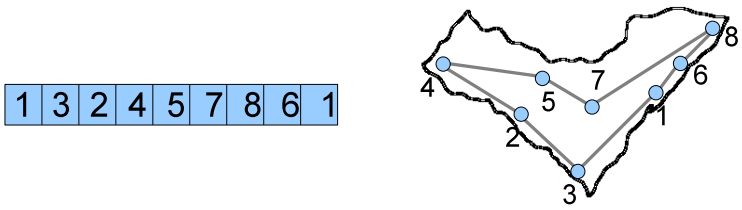

Figura 2.2: Representação vetorial de uma solução/cromossomo para o problema do caixeiro viajante, onde o vetor indica a sequência de nós a visitar.

1	3	2	4	5	7	8	6
3	2	4	5	7	8	6	1

Figura 2.3: Representação matricial de uma solução/cromossomo para o problema do caixeiro viajante onde a matriz indica a sequência de arestas a percorrer.

Representação por Tipos Abstratos de Dados

Uma forma de representação ainda pouco explorada é a representação por meio de tipos de dados abstratos, onde cada solução é codificada em um objeto[3]. Uma implementação desse tipo de algoritmo genético é o GAADT [4][17] que compartilha alguns dos seus princípios com modelos de algoritmos genéticos tradicionais, diferenciando-se principalmente pelas seguintes características: Todo problema tem uma função de codificação que mapeia todos os seus possíveis resultados em uma representação adotada para seus cromossomos; Todo cromossomo possui uma função de avaliação que mede o quão apta está aquela solução. Um trabalho recente[18], por exemplo, aplica o GAADT na geração de mapas em jogos de RPG.

2.1.3 Etapas de um Algoritmo Genético

Geração da População Inicial

A etapa de geração da população inicial é onde são criadas soluções com características aleatórias e é formada uma população inicial com um dado número destas soluções. A partir desta população inicia-se o processo de seleção natural e evolução, ou seja, otimização destas soluções. Em alguns problemas, a população inicial pode ser um parâmetro de entrada.

[3] Entenda-se por objeto as estruturas das linguagens orientadas a objeto, como classes ou structs.
[4] Genetic Algorithm based on Abstract Data Type

Função de Avaliação

Uma das etapas mais importantes do algoritmo genético, a função de avaliação é responsável pelo cálculo da aptidão da solução. Em problemas de otimização numérica, pode-se usar a própria função que se deseja otimizar como função de avaliação, em outros problemas é necessário que se elabore uma função capaz de traduzir os elementos do problema de forma a aferir a qualidade da solução encontrada. Portanto, de certa maneira, a função de avaliação deve ser capaz de representar o ambiente ao qual os indivíduos precisam se adaptar.

Condição de Parada

De maneira geral, pode-se conseguir soluções cada vez melhores a medida que mais ciclos se passam. É conhecido que, com um modelo elitista de AG, a medida em que o número de ciclos tende ao infinito, a solução encontrada tende à solução ótima [19]. No entanto, em algumas situações, torna-se necessário que a condição de parada leve em conta que, após ter melhorado, por muitos ciclos, a solução inicial, o algoritmo pode não atingir uma melhor solução dentro de um tempo prático pelo fato de estar convergindo muito lentamente.

Para resolver esse problema, na maioria dos algoritmos de busca de soluções, pode-se utilizar o seguinte como condição de parada: atingir um solução tão boa que não se altere após n ciclos OU alcançar m ciclos, onde m e n são parâmetros experimentalmente definidos.

Seleção

Esta é a etapa responsável pela escolha das soluções mais aptas para formar a próxima população. Nesta etapa, as soluções são ordenadas de acordo com suas aptidões, previamente aferidas pela função de avaliação, em seguida, as boas soluções são submetidas aos operadores de cruzamento. Em alguns problemas, a etapa de seleção também é responsável pela exclusão de soluções inválidas.

Operadores Genéticos

Os operadores genéticos são responsáveis pela modificação da solução existente. Essa modificação pode ter um efeito positivo ou negativo ao fim do ciclo, o que será contabilizado pela função de avaliação.

- Operador de Cruzamento:

  A depender do problema e da forma como a solução está sendo representada, ou seja, da estrutura do cromossomo artificial, várias formas de cruzamento são possíveis. As duas técnicas mais simples são a de cruzamento de um ponto e cruzamento de dois pontos, mostradas na Figura 2.4.

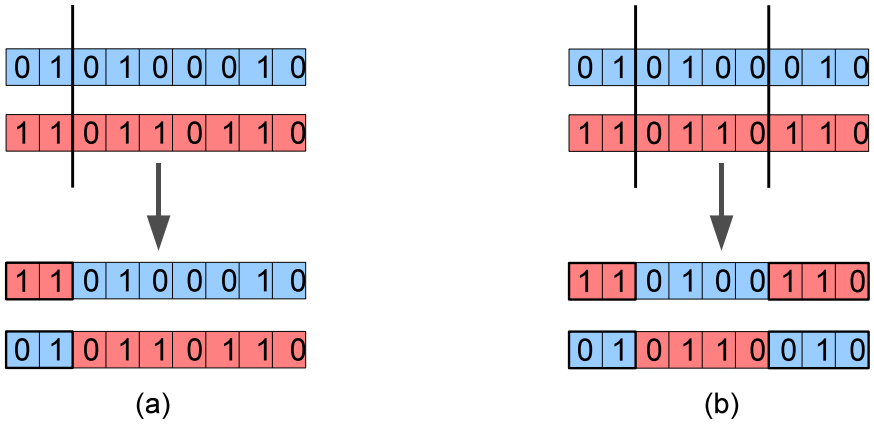

Figura 2.4: (a) Cruzamento de um ponto e (b) Cruzamento de dois pontos em um cromossomo de representação por meio de vetor binário.

O operador de cruzamento é responsável por propagar material genético de uma geração para outra, portanto, é de se esperar que as soluções seguintes sejam parecidas e portanto mantenham as boas características de suas soluções pais. Dessa forma, o operador de cruzamento contribui, em conjunto com a função de avaliação, para a convergência da aproximação.

- Operador de Mutação:

  O funcionamento do operador de mutação consiste em alterar cada característica da solução de acordo com uma dada probabilidade, introduzindo novo material genético na população. Para reproduzir bem o comportamento de uma mutação natural, essa probabilidade é geralmente muito baixa.

  De acordo com a representação da solução, este operador também pode assumir várias formas, numa representação por meio de vetor binário, por exemplo, a mutação consiste em inverter um bit, conforme Figura 2.5. Note que, por introduzir novo material genético na população, esse operador tem a capacidade de contornar o problema de máximo ou mínimo local encontrado em outros algoritmos como o K-means.

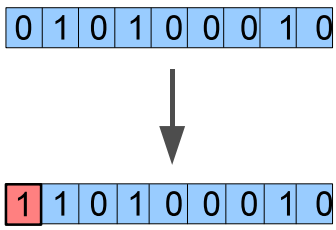

Figura 2.5: Mutação em um cromossomo de representação por meio de vetor binário.

É importante salientar que, por vezes, as soluções resultantes desses operadores genéticos são inválidas para o problema em questão e assim se faz necessário que estas sejam descartadas durante a etapa de seleção.

Substituição

Nesta etapa, a nova população, isto é, conjunto de soluções, é formada e submetida a função de avaliação para o início do novo ciclo.

2.2 Sobre a Tarefa de Agrupamento de Dados

A tarefa de agrupamento consiste em determinar um conjunto finito de categorias (grupos) para descrever um certo conjunto de dados de acordo com a semelhança entre seus objetos. A equação abaixo nos diz quantos grupos de k objetos são possíveis dentro de um conjunto de dados contendo n objetos.

$$N(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\left(\frac{k}{i}\right)(k-i)^n \tag{2.1}$$

Se tomarmos, por exemplo, $n = 25$ e $k = 5$, temos que $N(n,k) = 2.436.648.974.110.751$.

Esta tarefa é considerada um problema NP-Hard, [5] [6]e por isso tem se procurado por algoritmos de otimização eficientes que possam encontrar soluções próximas da solução ótima em tempo polinomial.

Um algoritmo largamente utilizado para a tarefa de agrupamento é o K-means. Em sua forma mais comum [20], algumas vezes referida por Algoritmo de Lloyd, um conjunto de elementos, usualmente vetores, é particionado de forma que cada elemento é atribuído à partição, ou grupo, de centróide mais próximo. A cada iteração do algoritmo, os centróides são recalculados de acordo com os elementos presentes no grupo e em seguida todos os elementos são realocados para a partição cujo o novo centróide se encontra mais próximo.

Em [3], uma série de experimentos com diversas bases de dados reais e artificiais é realizada com o intuito de comparar a eficiência do K-means com o AG para a tarefa de agrupamento. Os resultados mostraram que a eficiência do AG, no que diz respeito ao valor de aptidão das soluções obtidas, é significativamente superior.

Portanto, os algoritmos evolucionários [21], especialmente os algoritmos genéticos têm sido muito utilizados para agrupamento [4, 3, 5, 6], e detêm nossa atenção justamente por serem reconhecidamente bem sucedidos no cenário onde é necessária uma convergência

[5]Um problema é considerado NP-Hard quando pode-se assumir que sua complexidade é, pelo menos, tão grande quanto a do problema NP mais complexo.

[6]NP é o conjunto de problemas de decisão onde as instâncias para as quais a resposta é sim podem ser reconhecidas em tempo polinomial por uma máquina de Turing não determinística.

em tempo aceitável para soluções próximas da otimalidade em grandes espaços de busca [3].

Várias formas de representar a solução em um AG para um problema de agrupamento são utilizadas. Em [3] é utilizado um vetor de número reais, representados por variáveis de ponto flutuante, indicando o centróide de cada grupo. Em [4] uma forma relativamente nova de representar a solução foi utilizada, chamada de LLE[7], codificação por encadeamento linear, na sigla em inglês, que utiliza listas encadeadas para representar cada grupo, de forma que seus tamanhos possam ser heterogêneos. Em [6] e [5] o problema de formação de células de fabricação é abordado utilizando uma representação matricial.

Os operadores genéticos, como em todo AG, aqui terão papéis fundamentais. Em agrupamento, o operador de cruzamento deve ser capaz de promover a recombinação das soluções sem provocar soluções resultantes inválidas, ou pelo menos, provocando um pequeno número destas. Essa ressalva pode ser trivial de se tratar em algumas formas de representação, mas tem se mostrado desafiadora para a representação por meio de tipos abstratos de dados utilizada neste trabalho. Já a mutação, em agrupamento, acarretará na troca de um objeto de um grupo por um objeto de um outro.

As funções de avaliação mais comumente utilizadas na literatura são baseadas na distância entre os centróides de cada grupo, ou na distância média de um objeto a seu centróide. É necessário ressaltar que esse tipo de avaliação, simples como posto, traz alguns problemas, como por exemplo, uma solução onde cada objeto compõe um grupo pode ser superavaliada, da mesma forma que uma solução com um único grupo também corre o mesmo risco. Algumas abordagens [4] tem tratado desse problema utilizando-se de AGs multiobjetivos, onde mais de uma função de avaliação é utilizada.

Outra possibilidade também já abordada [21], e utilizada em nosso trabalho, é a composição de múltiplos objetivos em uma única função de avaliação, balanceando cada componente através de coeficientes, chamados de peso. Um problema dessa solução é que os pesos precisam ser definidos de antemão, o que pode requerer uma exaustiva experimentação.

[7] Linear Linkage Encoding

Capítulo 3

Abordagem Proposta

Para resolver o problema de agrupamento descrito na seção 2.2, propomos um Algoritmo Genético inspirado no modelo original proposto por Holland [9]. Esta implementação, que chamamos de AGrupos, utiliza uma representação das soluções por meio de tipos abstratos de dados e uma função de avaliação multiobjetivo composta.

O AGrupos foi implementado utilizando a linguagem de programação C++ com o framework Qt. Nesta implementação, fez-se uso deste framework por fornecer estruturas de dados, como listas, conjuntos e strings, além de avançados métodos de gerenciamento das mesmas, aumentando significativamente a produtividade da programação em linguagem C++.

Neste capítulo, iremos descrever com detalhes os elementos básicos do AGrupos, bem como apresentaremos um experimento com o intuito de avaliar a aplicabilidade dos AGs para o problema descrito, bem como formas de otimizá-lo.

3.1 O Problema

Mais especificamente, o problema que o AGrupos se propõe a resolver, como estudo de caso do uso de AGs para a tarefa de agrupamento, é o da formação de grupos em disciplinas de cursos de ensino a distância. Neste cenário, assumimos que um professor ou tutor deseje descobrir a melhor maneira de formar grupos numa determinada turma para a execução de uma atividade coletiva.

Assim, esperamos que a solução encontrada pelo agrupos preserve aspectos como a homogeneidade entre os grupos, e leve em consideração a coesão sociométrica de cada grupo[1], e a motivação dos alunos. Tais exigências podem ser entendidas como o ambiente ao qual a solução deverá adaptar-se e, portanto, são modeladas, no AGrupos, pela função de avaliação.

[1] Neste problema, entendemos por coesão sociométrica de um grupo a soma das susceptibilidades de cada aluno formar grupos com seus páreos do grupo.

3.2 Representação das Soluções

No problema abordado, o cromossomo, ou seja, a solução, é basicamente um conjunto de conjuntos de k alunos, onde k é um parâmetro de entrada; isto é, uma possível maneira de agrupar a turma. A Figura 3.1 exprime bem a ideia.

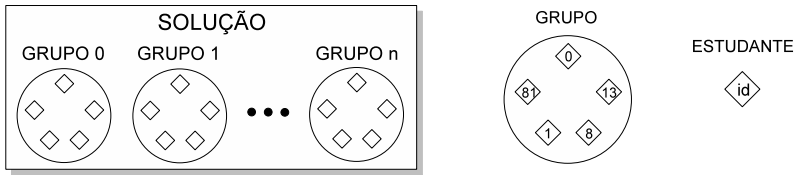

Figura 3.1: Representação da solução como um conjunto de grupos de alunos.

No AGrupos, essa estrutura foi implementada da seguinte forma:

- A entidade Estudante foi mapeada em uma classe, cujo cabeçalho apresentamos no Código A.1, que possui os atributos Id e Motivação, que pode ter o valor 0, para desmotivado, 1 para neutro, e 2 para motivado.
- A entidade Grupo foi mapeada em uma classe, cujo cabeçalho apresentamos no Código A.2, que possui como seus atributos uma estrutura de dados do tipo lista, mais precisamente uma QList, de k ponteiros para Estudantes e os inteiros *totalCohesion* que representa a coesão sociométrica, e *fitness* que armazena a aptidão do grupo.
- A entidade Solução também foi mapeada em uma classe (ver Código A.3) e possui como atributos uma QList de n ponteiros para Grupos e um inteiro *fitness* que armazena a aptidão da solução.
- O número total de estudantes numa solução é sempre $t = n \times k$, e portanto, os ids dos estudantes variam de 0 a $k-1$ sem repetição, uma vez que, em nosso cenário, os estudantes só podem estar em um grupo de cada vez.

No entanto, como vimos no capítulo 2, um dos grandes trunfos de um AG é sua capacidade de trabalhar paralelamente com várias soluções, as quais formam uma população, que no AGrupos representamos como uma QList de Soluções, conforme a Figura 3.2.

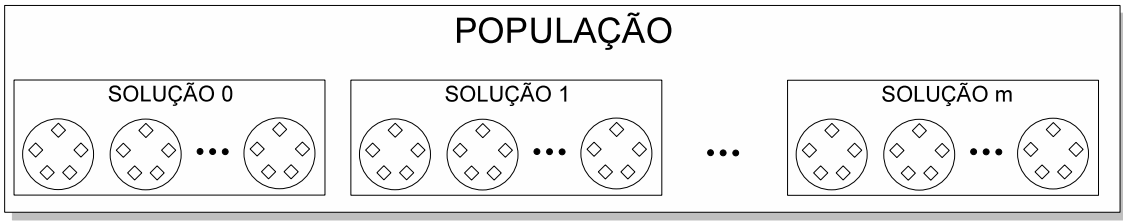

Figura 3.2: Representação da população.

3.3 Funcionamento do AGrupos

Como todo Algoritmo Genético, o AGrupos segue uma fluxo composto basicamente por: Geração de uma população inicial aleatória, avaliação da aptidão das soluções da população, seleção das melhores soluções, aplicação dos operadores genéticos e substituição da população antiga pela nova tal como descrito no fluxograma da Figura 3.3. Nesta seção, iremos descrever com detalhe sobre a implementação de cada uma dessas etapas do AGrupos.

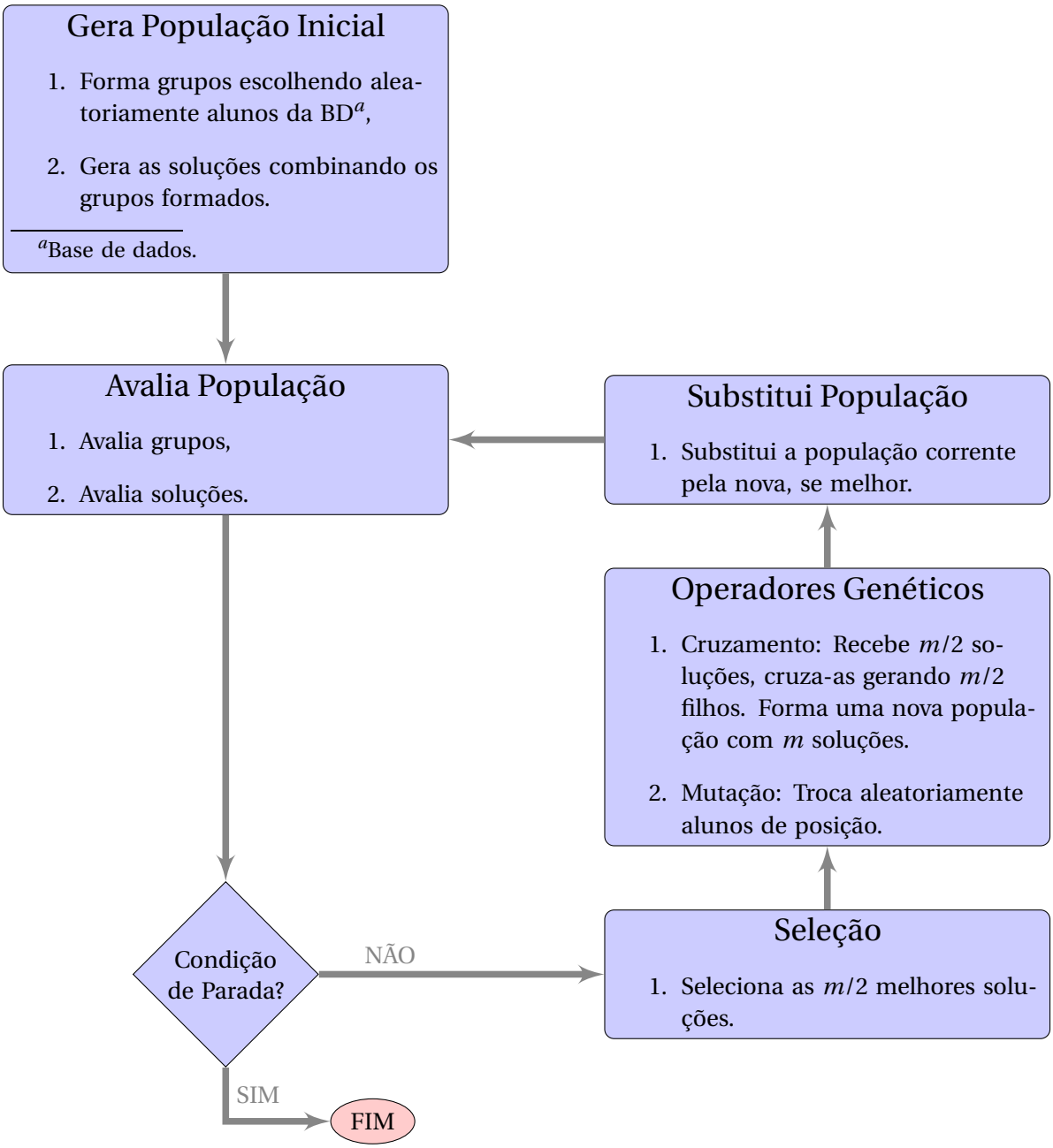

Figura 3.3: Fluxograma do AGrupos

3.3.1 Geração da População Inicial

A geração da população inicial no AGrupos consiste em, para cada uma das m soluções, ler os alunos da base de dados (BD) descrita no Apêndice B e formar n grupos de k estudantes. Ao formar cada grupo, é sorteado k vezes qual dos t estudantes comporão o novo grupo, ao fim, o grupo é incluído na solução, que quando completa, é incluída na população. O pseudo-código do algoritmo 2 resume bem o funcionamento dessa etapa cuja implementação pode ser vista no Código A.4.

Algoritmo 2 Geração da População Inicial

```
1: Criar uma nova população vazia
2: for i < m do
3:     Copiar estudantes da BD para uma lista auxiliar
4:     Criar uma nova solução
5:     for j < n do
6:         Criar um novo grupo
7:         for l < k do
8:             Sortear um estudante da lista auxiliar
9:             Retirar estudante da lista auxiliar e colocar no novo grupo.
10:            l = l + 1
11:        end for
12:        Colocar novo grupo na nova solução
13:        j = j+1
14:    end for
15:    Colocar nova solução na nova população.
16:    i = i+1
17: end for
```

3.3.2 Função de Avaliação

Se observarmos bem o problema definido na Seção 3.1, notaremos que é composto de múltiplos objetivos. Por um lado, precisamos valorizar homogeneidade entre os grupos de uma solução, por outro, maximizar a coesão sociométrica de cada grupo e ainda garantir que estes sejam compostos de uma mistura balanceada de alunos motivados ou neutros, e desmotivados.

Para tanto, no AGrupos, propomos uma composição desses objetivos numa única função de avaliação, controlando os diversos aspectos através dos coeficientes aqui denominados "pesos". Para efeito de compreensão, a função de avaliação do AGrupos divide-se, como visto no fluxograma da Figura 3.3, em duas etapas: A avaliação dos grupos e a avaliação das soluções.

Avaliação dos Grupos

A aptidão de cada grupo é medida levando em conta o nível de motivação dos alunos e a coesão sociométrica, que representa o quanto cada estudante do grupo gostaria de formar grupo com cada um dos outros colegas do grupo em que se encontra. Dessa forma, é valorizada a vontade, do estudante, de estar naquele grupo.

$$f = w_{sc}\sigma + w_{sm}\sum_{0}^{k-1} m_s \tag{3.1}$$

Onde:
f = a aptidão do grupo.
w_{sc} = o peso dado à coesão sociométrica.
σ = a coesão sociométrica.
w_{sm} = o peso dado à motivação do estudante.
m_s = a motivação do estudante.

A equação 3.1 expressa o cálculo da aptidão de um grupo. No AGrupos, para obter a informação sobre a coesão sociométrica, foi utilizado a soma do número de visitas que cada estudante realizou ao blog de seus colegas de grupos. Esta aproximação foi tomada para garantir que o processo de formação de grupos possa ser feita de forma automática baseando-se apenas nos dados presentes em base de dados de sistemas de ensino a distância.

Avaliação da Solução

Após terem sido avaliados todos os grupos pertinentes à solução, passamos a calcular as grandezas relacionadas à solução em si e então compor a função de avaliação da solução:

$$F = \frac{\sum_{0}^{n-1} f}{(w_h D + 1)} \times \frac{G_m}{k} \tag{3.2}$$

Onde:
F = a aptidão da solução.
w_h = o peso dado à homogeneidade da solução.
D = a maior diferença entre os f dos grupos dessa solução.
G_m = quantidade de grupos que possuem mais da metade de seus estudantes não desmotivados.
m = quantidade de soluções em uma população.
n = quantidade de grupos em uma solução.
k = quantidade de estudantes em um grupo.

Note que, na equação 3.2, a primeira parte da fração está relacionada com a homogenei-

dade dos grupos, pois o seu valor será menor quanto maior for a diferença entre as aptidões do pior e do melhor grupo da solução. Já a segunda parte (G_m/k), garante que a avaliação da solução será maior se a maior parte dos grupos forem compostos por mais da metade de estudantes não desmotivados. Somamos uma unidade ao denominador para garantir que não haverá inconsistência quando D for nulo.

Observe que esta última parte da fração trabalha em conjunto com a equação 3.1, que de forma isolada, valoriza os grupos que possuem a maior quantidade de alunos motivados ou neutros.

3.3.3 Condição de Parada

O laço principal do AGrupos, responsável pelo ciclo observado no fluxograma da Figura 3.3, pára quando não se encontra uma melhor solução mesmo após δ ciclos desde a última melhoria ou quando se atinge um determinado número máximo μ de ciclos.

Os parâmetros δ e μ são definidos experimentalmente observando o comportamento do algoritmo, através das curvas de convergência que mostraremos no capítulo 4, com valores inicialmente altos e em seguida diminuindo até valores próximos dos tamanhos dos platôs. De modo geral, os AGs tendem a convergir rapidamente para algum máximo local[2] onde ficam presos durante um determinado número de ciclos até que se encontre uma melhor solução. O parâmetro δ, portanto, pode ser interpretado como o grau tolerância a esse efeito.

3.3.4 Seleção

Nesta etapa, as m soluções são ordenadas de acordo com sua aptidão dentro da população de forma ascendente. Em seguida, a segunda metade, ou seja, as $m/2$ melhores soluções são selecionadas e guardadas em uma população temporária, a qual é submetida ao operador de cruzamento.

3.3.5 Operador de Cruzamento

Como vimos, em todo AG, a essência de um operador de cruzamento é propagar material genético de uma população para outra, ou seja, manter nas próximas populações as boas características das boas soluções. Em um problema de agrupamento existem várias maneiras de atingir esse objetivo, no AGrupos implementamos uma variação do Cruzamento Orientado a Grupos, COG [22].

O processo pode ser melhor compreendido por meio da Figura 3.4. Na Etapa 1, são escolhidas, ao acaso, duas soluções das $m/2$ resultantes da seleção. Essas duas soluções, terão suas características combinadas, para tanto, na Etapa 2, dois grupos são selecionados também ao acaso, um de cada solução. Na Etapa 3, guardamos em dois vetores as diferenças

[2] O máximo local pode ser também o máximo global.

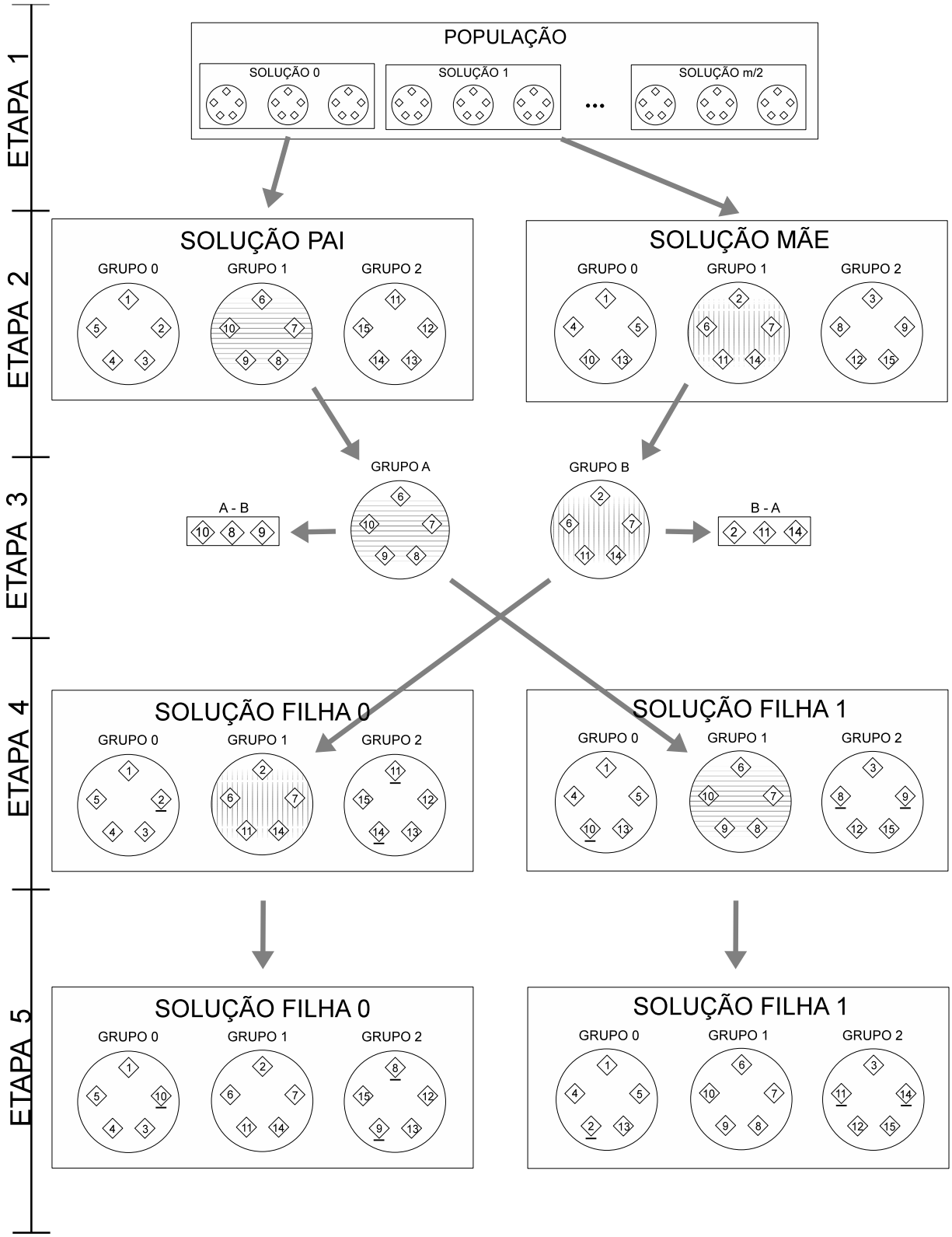

Figura 3.4: Cruzamento Orientado a Grupos tal como implementado no AGrupos.

entre os dois grupos selecionados, esses dois vetores servirão para guiar as mutações corretivas que se farão necessárias ao final do processo. Em seguida, na Etapa 4, os dois grupos trocam de lugar, o que gera soluções inválidas, pois alguns estudantes passam a aparecer duas vezes e outros desaparecem da solução, nos obrigando a efetuar mutações corretivas da Etapa 5.

Para efetuar as correções, primeiramente buscamos o conteúdo do vetor B-A entre os grupos nativos da Solução Filha 0 (Estudantes sublinhados na Etapa 4 da Figura 3.4) e, ao encontrar, muta-se para um dos estudantes do vetor A-B. De forma análoga corrigimos a Solução Filha 1, resultando em duas novas soluções válidas, as quais são concatenadas ao fim da população original.

Todo o processo é repetido para cada par das $m/2$ soluções, ou seja, $m/4$ vezes, resultando em uma nova população de tamanho m.

Se olharmos para codificação da população, representada na Figura 3.2, podemos entender a estrutura como um grande vetor de estudantes, onde a cada k elementos temos um grupo e a cada $k \times n$ temos uma solução. Dessa forma, podemos entender o COG como um cruzamento de dois pontos, representado na Figura 2.4, onde o primeiro ponto situa-se no início do grupo a ser cruzado e o segundo ao fim do mesmo. No entanto há de se ressaltar que as mutações corretivas, feitas ao final do processo, é que tornam único esse tipo de cruzamento.

Operador de Mutação

No AGrupos, a etapa de mutação consiste em permutar estudantes de diferentes grupos. Para tanto, o algoritmo percorre cada par de grupos na solução, selecionando cada par possível de estudantes e sorteia se os mesmos serão permutados ou não, de acordo com a taxa de mutação pré-definida. A seguir, no algoritmo 3, apresentamos pseudo-código correspondente ao operador de mutação que implementamos.

Substituição

A etapa substituição consiste em verificar se a nova população possui alguma solução melhor do que a corrente, se sim, a população corrente é substituída pela nova.

Algoritmo 3 Operador de Mutação

```
1: Recebe uma população
2: for i < m do
3:     for j < n do
4:         for l < n do
5:             if j != l then
6:                 Grupo1 = Grupo em j
7:                 Grupo2 = Grupo em i
8:                 for p < k do
9:                     if sorteio(de 0 a 100) < ProbabilidadeDeMutacao then
10:                        Troca estudante p do Grupo1 pelo p do Grupo2
11:                    end if
12:                    p = p + 1
13:                end for
14:            end if
15:        end for
16:        j = j + 1
17:    end for
18:    m = m + 1
19: end for
```

Capítulo 4

Avaliação da Proposta

Utilizando uma base de dados artificial, conforme descrita no Apêndice B realizamos um experimentos variando o parâmetro que regula a probabilidade de mutação. Para cada variação do parâmetro, executamos o programa 4 vezes, plotando, para cada uma delas, um gráfico que apresenta a variação da aptidão da melhor solução de cada população de acordo com os ciclos do AG.

Os parâmetros fixos durante todo o experimento foram: o peso dado à coesão sociométrica $w_sc = 0,5$, o peso dado à motivação do estudante $w_m = 0,5$, e o peso dado à homogeneidade dos grupos $w_h = 0,5$. Além destes, os parâmetros referentes à condição de parada, descritos na Subseção 3.3.3, foram definidos da seguinte forma: $\delta = 100$ e $\mu = 5000$.

Como podemos ver, por meio dos gráficos das figuras 4.1 – 4.10, de forma geral, o AGrupos apresenta uma convergência relativamente rápida com comportamento próximo de uma exponencial positiva.

É possível notar, através dos platôs visíveis nos gráficos, que o problema da otimalidade local ainda causa um certo atraso na evolução da aptidão em algumas execuções do algoritmo, enquanto em outras, como as da figura 4.9, ele quase não ocorre.

Para cada valor do parâmetro de mutação, medimos a taxa de convergência, que aproximamos pela razão entre a média das máximas aptidões atingidas e a média dos ciclos necessários para alcançá-las. Com isso plotamos o gráfico da figura 4.11 por meio do qual pudemos analisar a influência do nosso operador de mutação na convergência do algoritmo.

Aparentemente, como a taxa de convergência oscila durante os experimentos, apresentando um comportamento errático, o operador de mutação, simples como implementamos, não foi eficiente em realizar sua tarefa de evitar os máximos locais acelerando a convergência. O que nos leva a crer que apenas o operador de cruzamento e suas mutações corretivas estão contribuindo para a convergência no AGrupos.

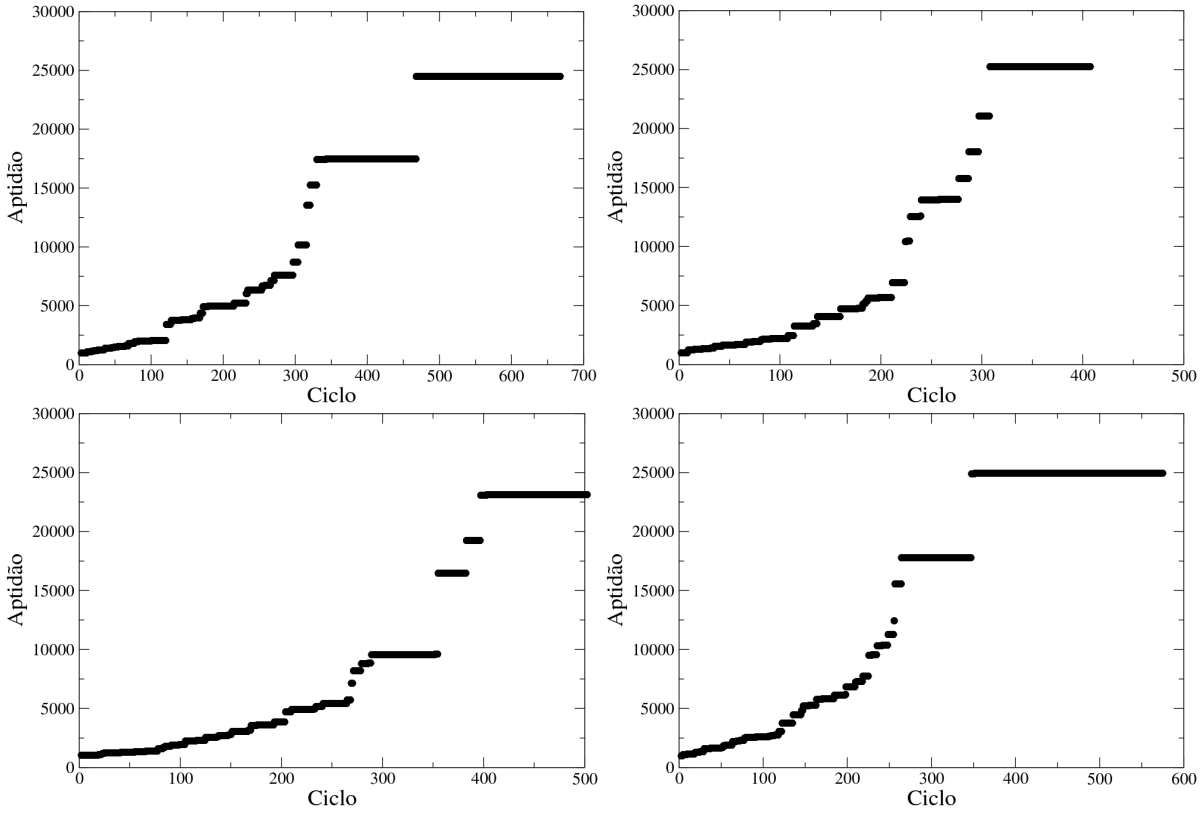

Figura 4.1: Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 0.01%.

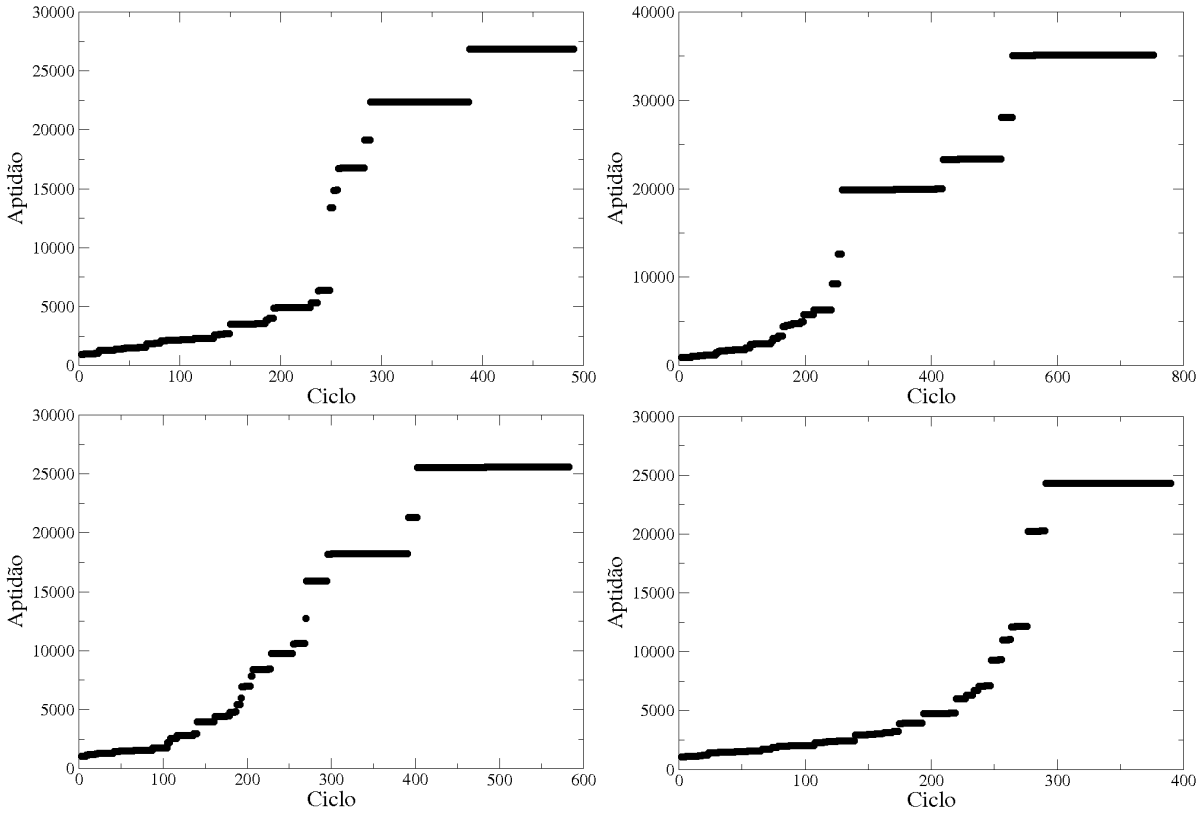

Figura 4.2: Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 1%.

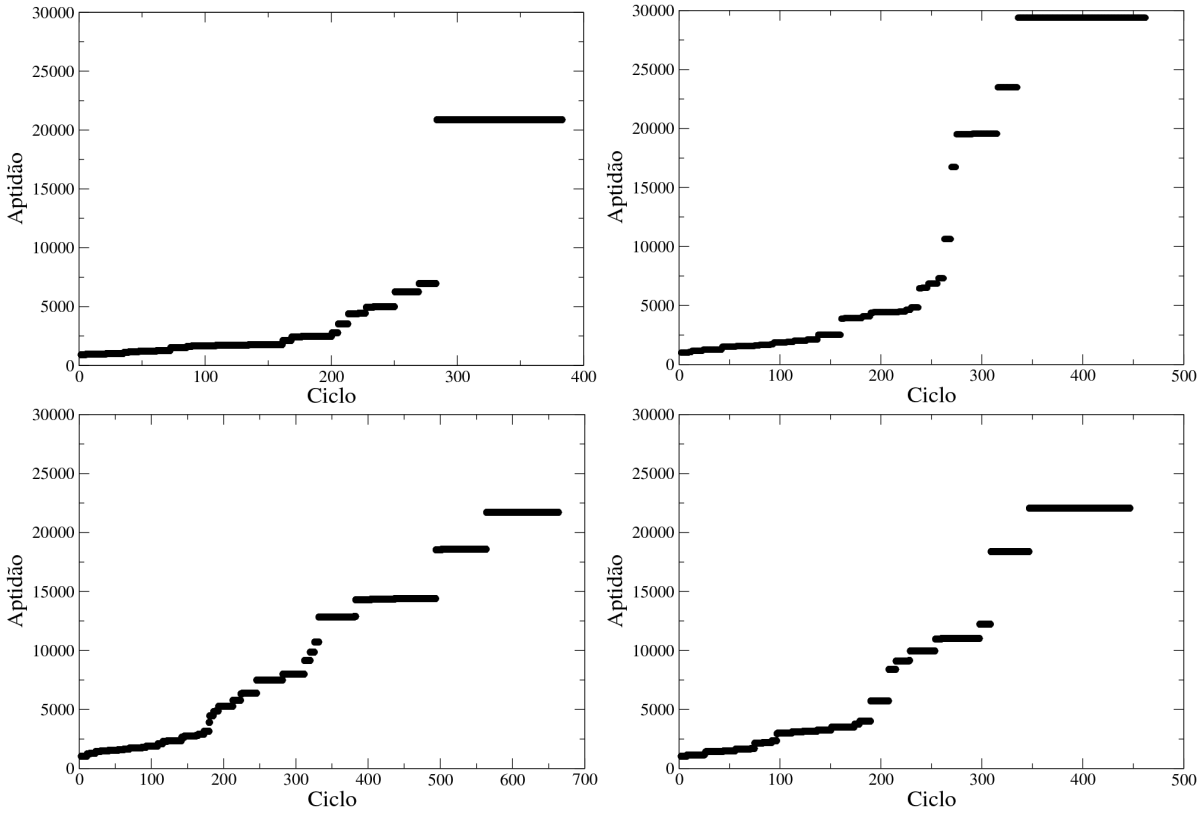

Figura 4.3: Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 4%.

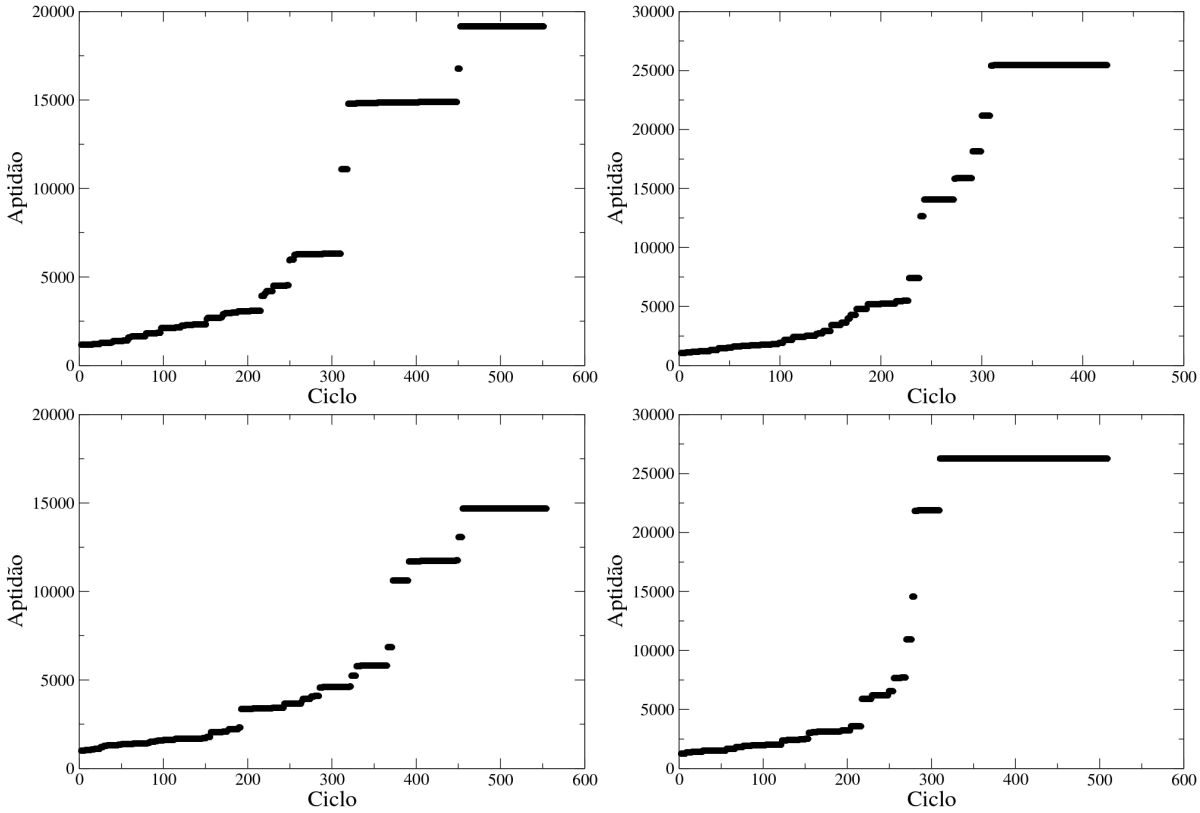

Figura 4.4: Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 9%.

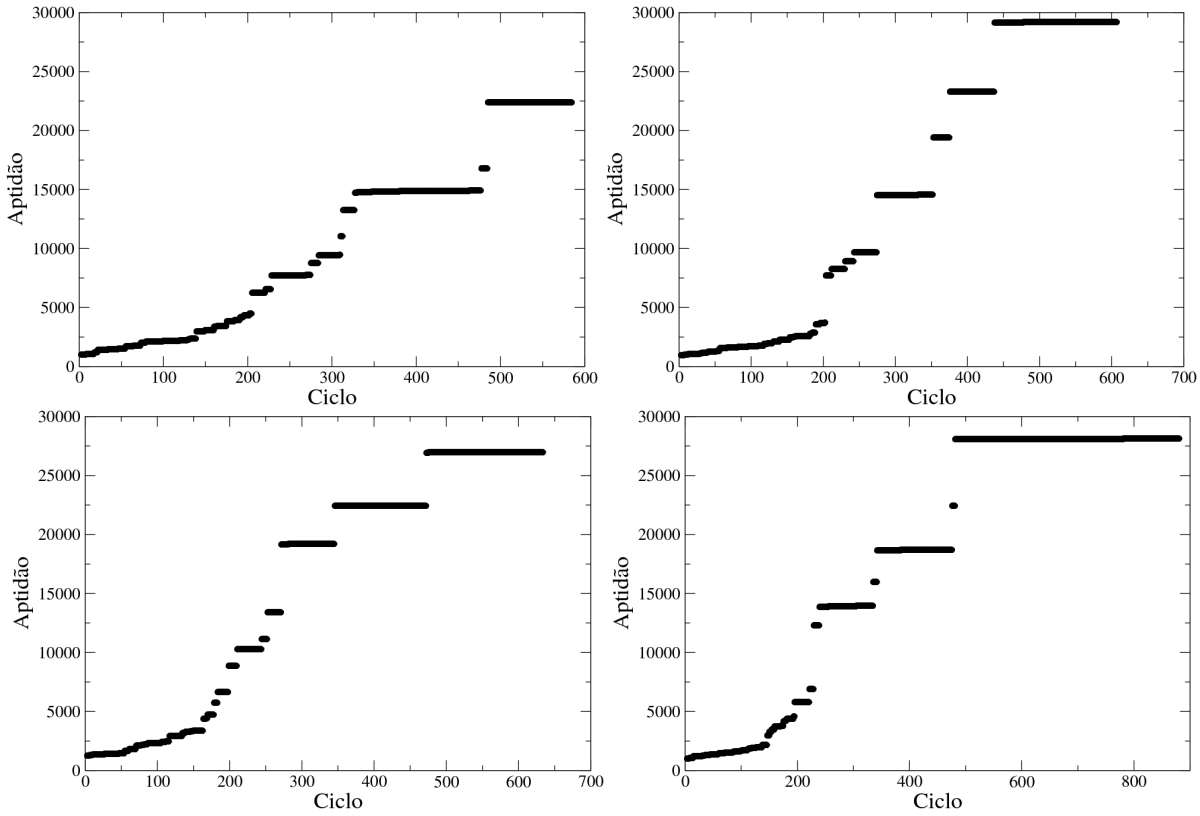

Figura 4.5: Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 15%.

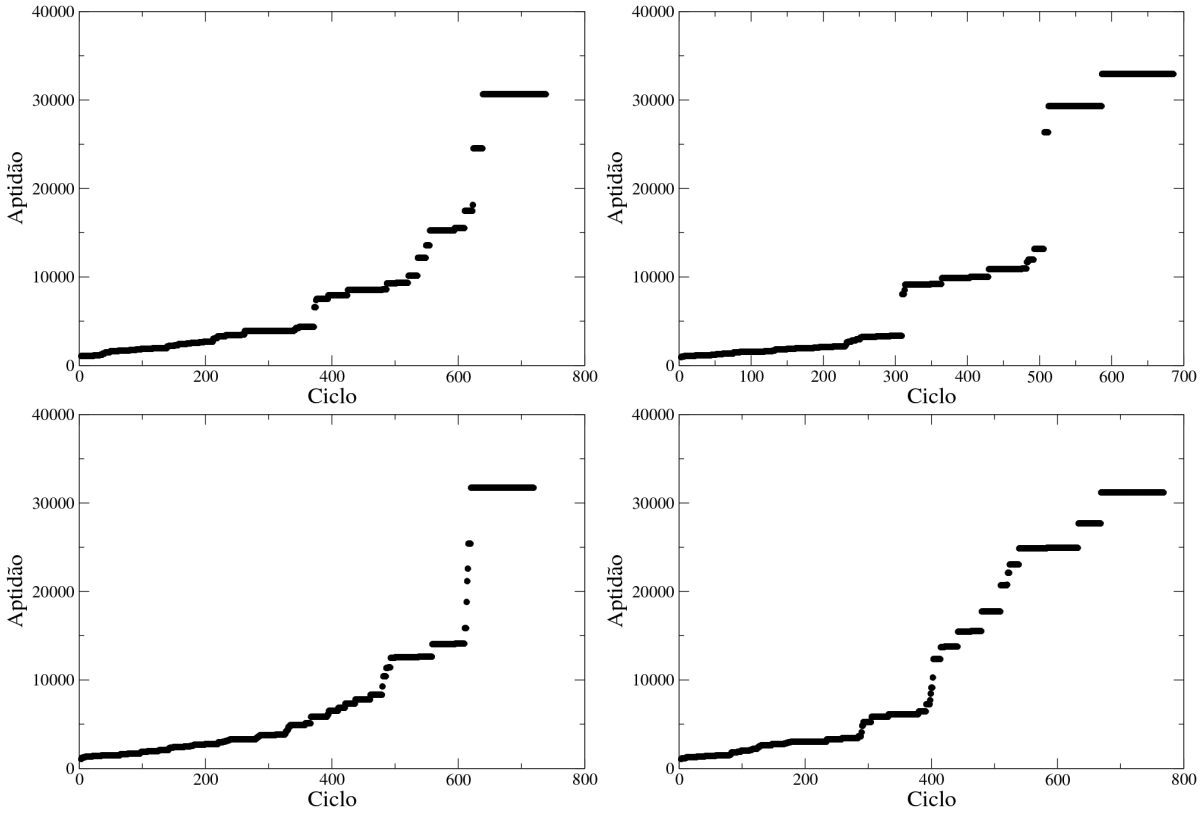

Figura 4.6: Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 25%.

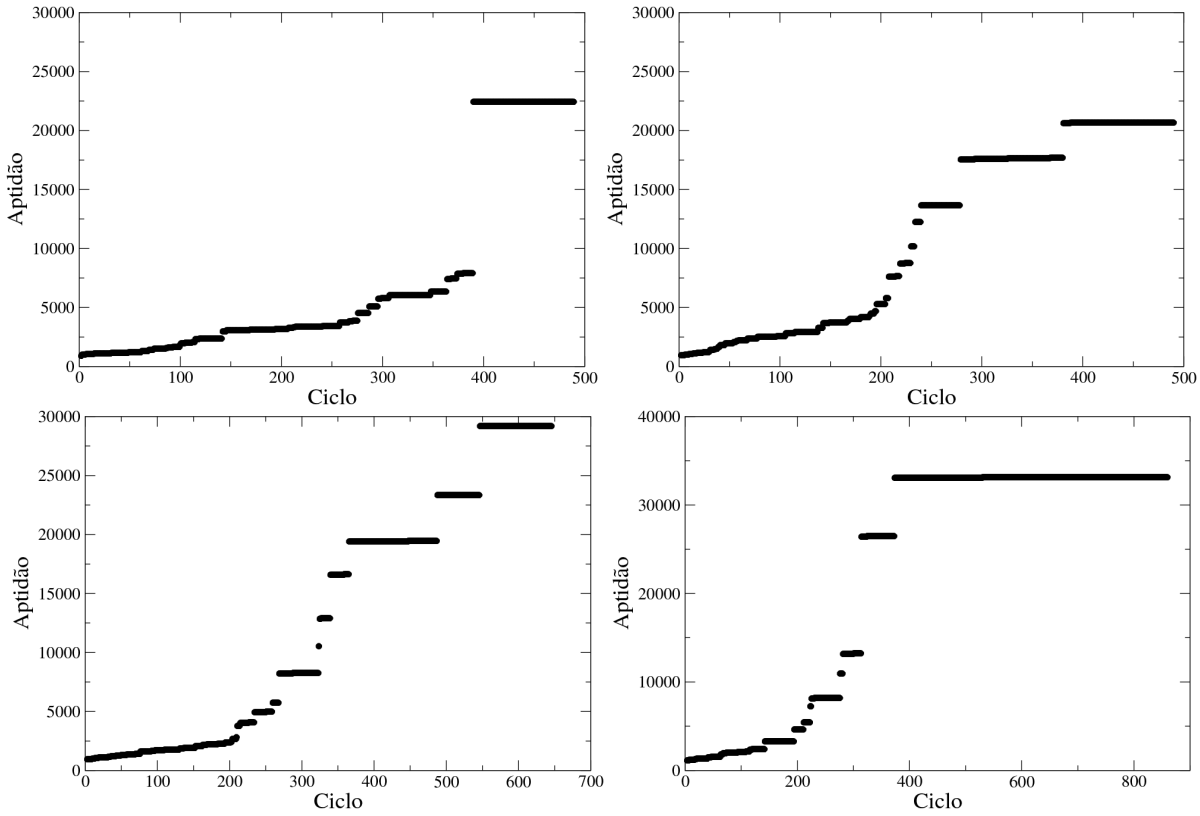

Figura 4.7: Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 30%.

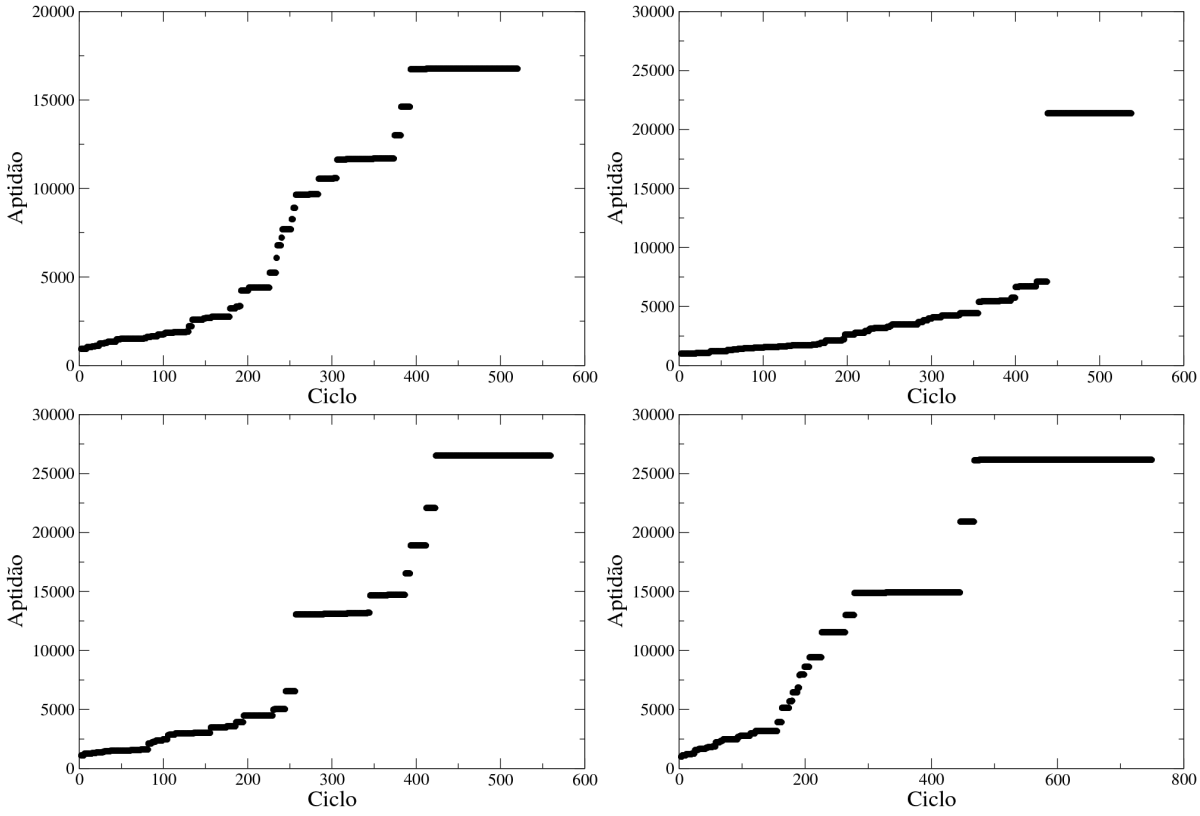

Figura 4.8: Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 40%.

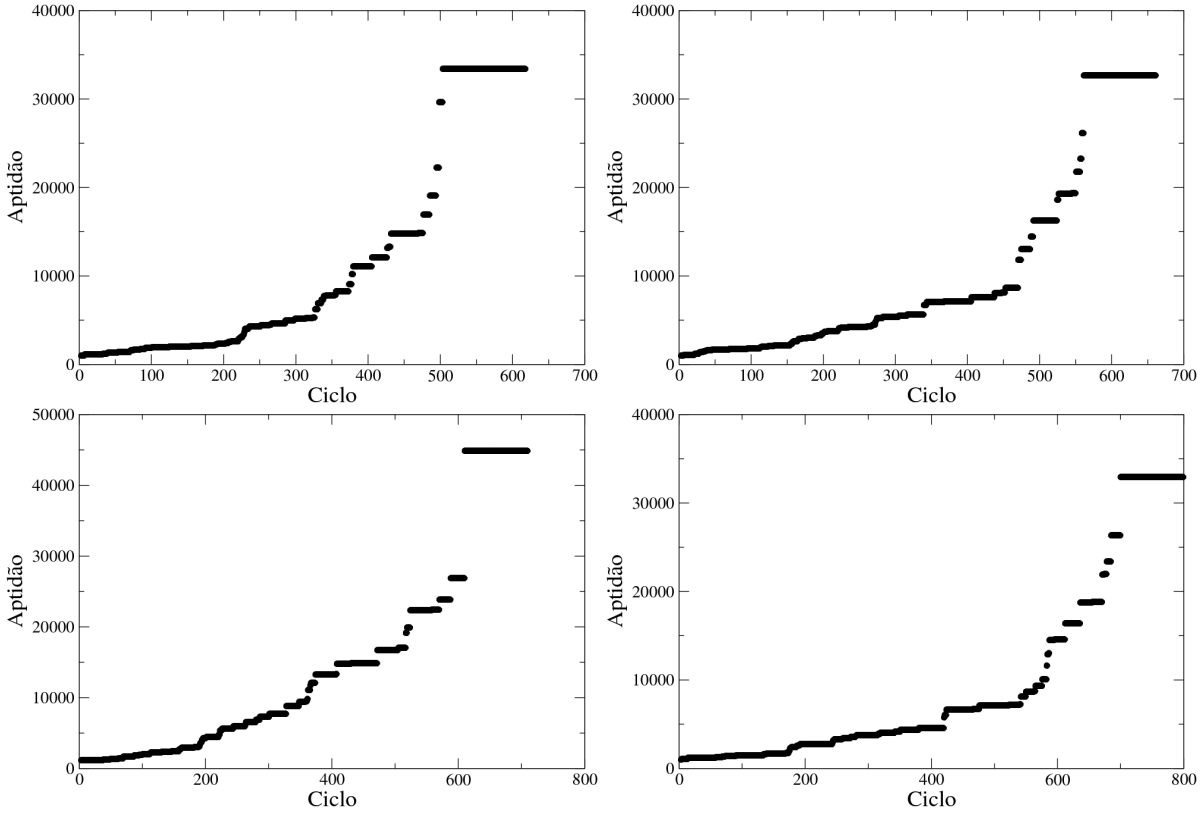

Figura 4.9: Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 50%.

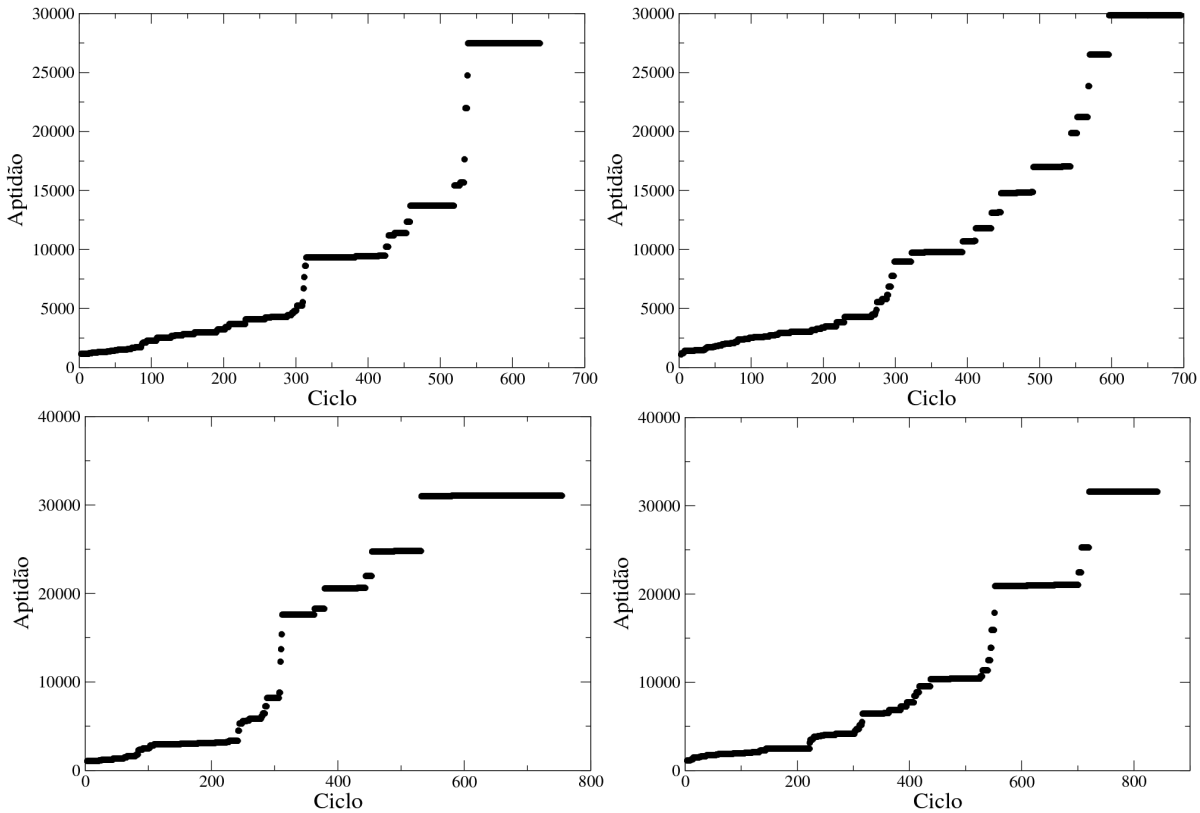

Figura 4.10: Evolução da aptidão de acordo com os ciclos. Prob. de Mutação = 60%.

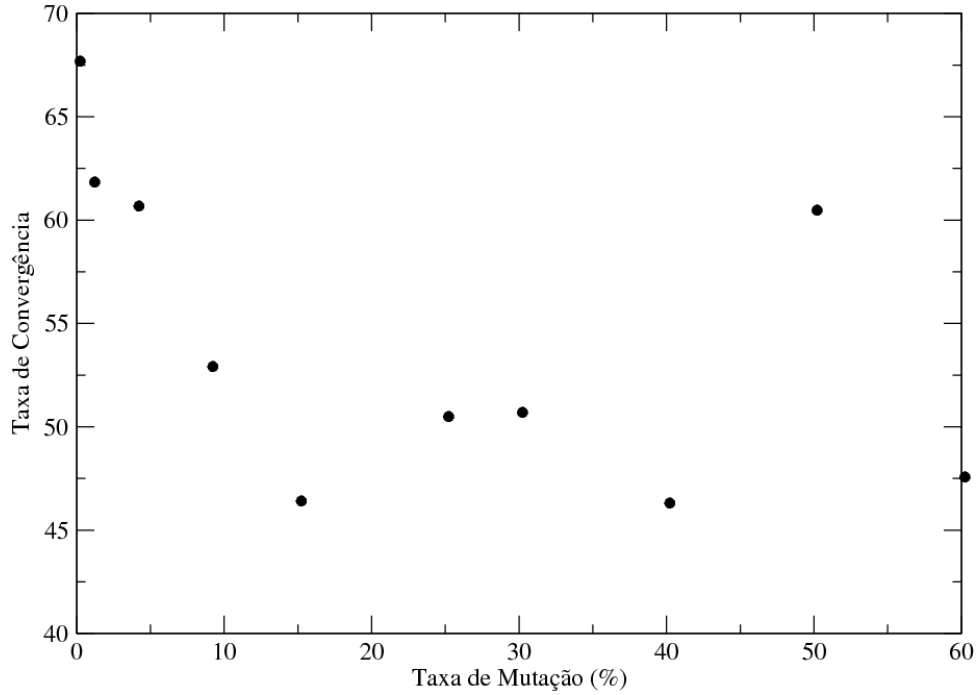

Figura 4.11: Variação da taxa de convergência da aptidão das soluções de acordo com a probabilidade de mutação.

Capítulo 5

Conclusão

Este trabalho se propôs a desenvolver e analisar uma implementação de um Algoritmo Genético para a tarefa de agrupamento de dados, particularmente, sobre dados provenientes de Ambientes Virtuais de Aprendizagem. Nesse contexto, desenvolveu-se o AGrupos, que se mostrou apto a realizar efetivamente a tarefa de formação de grupos de alunos numa turma de um curso de ensino a distância, conforme avaliação realizada no Capítulo 4.

A representação das soluções por meio de tipos abstratos de dados facilitou a implementação do AGrupos, garantindo a facilidade de expansão dessa representação, com o objetivo de incluir mais dados a respeito dos alunos ou dos grupos, uma vez que para tanto basta modificar a implementação das classes acrescentando atributos e métodos.

O Operador de Cruzamento Orientado a Grupos que implementamos foi efetivo na tarefa de propagar as boas características de uma solução para as populações seguintes, mantendo a convergência do algoritmo em bons níveis, conforme ilustrado pelo gráfico apresentado na Figura 4.11 e pelo comportamento das curvas dos gráficos nas Figuras 4.1 – 4.10.

No entanto, um de nossos anseios, a otimização do algoritmo por meio do estudo da influência do operador de mutação não foi possível pelo fato desse estudo ter nos mostrado que o operador, simples como implementamos, ou seja, baseado em permutações aleatórias de estudantes de um grupo para outro, sem basear-se em algum critério de avaliação do estudante, parece não contribuir de forma sensível para a convergência do algoritmo.

Outra limitação de nossa abordagem foi a forma como avaliamos um estudante, baseando-se apenas em sua motivação, um parâmetro de três valores, que, apesar de ser baseado em muitas outras informações presentes no AVA, fez com que trabalhássemos com apenas três categorias de alunos. Podemos inclusive inferir que esta simplificação tenha desvalorizado o operador de mutação, pois a cada troca, a chance de se formar grupos, e portanto soluções, diferentes do ponto de vista da motivação dos estudantes fica reduzida, uma vez que a quantidade de alunos de mesma motivação é relativamente alta em relação ao tamanho da base.

Em trabalhos futuros pretendemos sanar essas limitações, primeiramente desenvol-

vendo um operador de mutação mais inteligente, permutando alunos que não estejam contribuindo para o aumento da avaliação de seus grupos atuais, e em seguida, expandindo a representação da solução utilizando mais informações presentes nos bancos de dados dos AVAs, como por exemplo, o horário mais frequente de visita ao ambiente. Pretendemos ainda, explorar uma aplicação direta do AGrupos em Ambientes Virtuais de Aprendizagem, como o Moodle [7] e o Massayó [8], onde poderíamos incluí-lo na forma de um *plugin*, através do qual um professor ou tutor poderia agrupar suas turmas de forma automática e de acordo com parâmetros configuráveis.

Referências

[1] E. L. Passos and R. Goldsmith. *Data Mining: Um guia Prático*. Campus, 1 edition, 2005.

[2] S. Z. Selim and M. A. Ismail. K-means-type algorithms - a generalized convergence therorem and characterization of local optimality. *IEEE Trans. Patt. Analysis and Machine Intel.*, 6(1):81–87, 1984.

[3] U Maulik and S Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern Recognition*, 33(9):1455–1465, SEP 2000.

[4] E. E. Korkmaz. Multi-objective Genetic Algorithms for grouping problems. *Applied Intelligence*, 33(2):179–192, OCT 2010.

[5] Adnan Tariq, Iftikhar Hussain, and Abdul Ghafoor. A hybrid genetic algorithm for machine-part grouping. *Computers and Industrial Engineering*, 56(1):347–356, FEB 2009.

[6] T. Tunnukij and C. Hicks. An enhanced grouping genetic algorithm for solving the cell formation problem. *International Journal of Production Research*, 47(7):1989–2007, 2009.

[7] Martin Dougiamas. Moodle.org: open-source community-based tools for learning. `http://moodle.org`, 2011. [Online; acessado em Fevereiro de 2011].

[8] Ig Ibert Bittencourt, Evandro Costa, Marlos Silva, and Elvys Soares. A computational model for developing semantic web-based educational systems. *Knowledge-Based Systems*, 22(4):302 – 315, 2009. Artificial Intelligence (AI) in Blended Learning - (AI) in Blended Learning.

[9] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, April 1992.

[10] WC Zhong, J Liu, MZ Xue, and LC Jiao. A multiagent genetic algorithm for global numerical optimization. *IEEE Trans. Syst., Man, Cybern. B, Cybernetics*, 34(2):1128–1141, April 2004.

[11] C. W. Ahn and R. S. Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Trans. on Evolutionary Computation*, 6(6):566–579, December 2002.

[12] QF Zhang and YW Leung. An orthogonal genetic algorithm for multimedia multicast routing. *IEEE Trans. on Evolutionary Computation*, 3(1):53–62, April 1999.

[13] I.S. Oh, J.S. Lee, and B.R. Moon. Hybrid genetic algorithms for feature selection. *IEEE Trans. Patt. Analysis and Machine Intel.*, 26(11):1424–1437, November 2004.

[14] J. H. Yang and V Honavar. Feature subset selection using a genetic algorithm. *IEEE Inteligent Syst. and their Applications*, 13(2):44–49, March-April 1998.

[15] G. C. Onwubolu and M. Mutingi. A genetic algorithm approach to cellular manufacturing systems. *Computers and Industrial Engineering*, 39(1-2):125–144, February 2001.

[16] J. F. Goncalves and M. G. C. Resende. An evolutionary algorithm for manufacturing cell formation. *Computers and Industrial Engineering*, 47(2-3):247–273, November 2004.

[17] R. V. V. Lopes. A genetic algorithm based on abstract data type and its specification in z. *Computer Center, Federal University of Pernambuco*, 2003.

[18] L. F. B. Silva de Carvalho, H. C. Silva Neto, R. V. V. Lopes, and F. Paraguacu. An application of genetic algorithm based on abstract data type for the problem of generation of scenarios for electronic games. In *Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on*, volume 2, pages 526 –530, October 2010.

[19] D. Bhandari, C. A. Murthy, and S. K. Pal. Genetic algorithm with elitist model and its convergence. *International Journal of Pattern Recognition and Artificial Inteligence*, 10(6):731–747, SEP 1996.

[20] S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129 – 137, March 1982.

[21] E. R. Hruschka, R. J. G. B. Campello, A. A. Freitas, and A. C. P. L. F. de Carvalho. A survey of evolutionary algorithms for clustering. *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev*, 39(2):133–155, March 2009.

[22] Giselher Pankratz. A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum*, 27:21–41, 2005. 10.1007/s00291-004-0173-7.

Apêndice A

Código do AGrupos

Neste apêndice apresentamos alguns trechos de código do AGrupos com o objetivo de auxiliar a compreensão da implementação e dos algoritmos, portanto, ressaltamos que muitas linhas de código não relacionado diretamente com o algoritmo genético foram omitidas.

Código A.1: Cabeçalho da classe Estudante

```
1  class Student
2  {
3  public:
4      Student(int motivation, int id);
5      Student();
6      bool getMotivation();
7      int getId();
8  private:
9      int id;
10     bool motivation;
11 };
```

Código A.2: Cabeçalho da classe Grupo

```
1  class Group
2  {
3  public:
4      Group(const Group &group);
5      void addStudent(Student *student);
6      void calculateCohesions();
7      void setFitness(int fitness);
8      int getSocioCohesion();
9      int getFitness();
10     QList<Student*> getStudents();
```

```
11      QList<Student*> *getRealStudents();
12      QList<int> getStudentIds();
13
14  private:
15      int totalCohesion;
16      int fitness;
17      QList<Student*> students;
18  };
```

Código A.3: Cabeçalho da classe Solução

```
1   class Solution
2   {
3   public:
4       Solution();
5       Solution(const Solution &solution);
6       void addGroup(Group *group);
7       QList<Group*> getGroups();
8       QList<Group*> *getRealGroups();
9       void setFitness(int fitness);
10      int getFitness();
11      int greatestFitnessDifference();
12      bool operator<(Solution const & solution) const;
13  private:
14      QList<Group*> groups;
15      int fitness;
16  };
```

Código A.4: Função de Geração da População Inicial

```
1   QList<Solution*> generateInitialPopulation(){
2       //Generating a random population:
3       QList<Solution*> population;
4       for (int i = 0; i < populationSize; i++)
5       {
6           //Receiving students list from the PreProcessor:
7           PreProcessor::setStudentsQuantity(numberOfStudents);
8           QList<Student*> students = PreProcessor::getInstance()->getStudents();
9           //Generating a random Solution:
10          Solution *solution = new Solution();
11          for (int j = 0; j < solutionSize; j++)
```

```
12          {
13              //Generating a random Group:
14              Group *group = new Group(j);
15              for (int k = 0; k < groupSize; k++)
16              {
17                  //Pick a student randomly:
18                  int randomNumber = aleat(0,students.size() - 1);
19                  group->addStudent( students.takeAt(randomNumber) );
20              }
21              solution->addGroup(group);
22          }
23          population.append(solution);
24      }
25      return population;
26  }
```

Apêndice B

Base de Dados utilizada para avaliação do AGrupos

Para efeito de avaliação da nossa solução, geramos uma Base de Dados aleatória, composta de dois arquivos. O primeiro arquivo, do qual trechos são apresentados na Tabela B.1, contém uma linha para cada um dos estudantes e duas colunas, uma com o Id deste e outra com sua motivação, de 0 a 2. O segundo arquivo, com trechos apresentados na Tabela B.2, contém uma tabela que representa as visitas a blogs. Na primeira coluna temos o Id do visitante, na segunda, o Id do visitado e, finalmente, na terceira coluna, quantas vezes ocorreu esta visita.

Os parâmetros relativos ao tamanho da BD, para este experimento, foram definidos dessa forma: O tamanho da população $m = 20$, o tamanho da solução $n = 60$, e o tamanho do grupo $k = 5$.

Id do Estudante	Motivação
0	2
1	0
1	1
2	1
3	1
4	1
5	1
6	2
7	0
8	2
9	2
10	0
11	0
12	2
.	.
.	.
.	.
293	0
294	2
295	1
296	2
297	1
298	1
299	0

Tabela B.1: Trechos da tabela de estudantes.

Visitante	Visitado	Número de Visitas
0	1	2
0	2	10
0	3	2
0	4	10
0	5	8
0	6	0
0	7	12
0	8	3
0	9	9
0	10	19
0	11	12
0	12	7
0	13	1
0	14	12
0	15	0
0	16	1
0	17	6
.	.	.
.	.	.
.	.	.
0	286	15
0	287	3
0	288	8
0	289	11
0	290	7
0	291	15
0	292	20
0	293	4
0	294	14
0	295	16
0	296	7
0	297	17
0	298	14
0	299	9

Tabela B.2: Trecho da tabela de visitas a blogs onde se vê as visitas realizadas pelo estudante de Id 0. O padrão se repete para os outros 299 estudantes.