Prospecção sobre a Classificação de Demandas por Afinidades

Maria Isabella de Oliveira Novais [1], Murilo Fernandes Leobas[2], Bernardo Souza Ferreira da Silva[3], Andresa da Silva Lima[3], Paulo Sérgio Atavila Júnior[4], Marcos Balduino de Alvarenga[5]

[1]Graduanda em Engenharia Elétrica - IFTO. Bolsista do CNPq. e-mail: <maribella.novais@gmail.com>
[2]Graduando em Engenharia Elétrica - IFTO. Bolsista do IFTO. e-mail: <muriloleobas17@gmail.com>
[3]Graduandos em Engenharia Elétrica - IFTO. e-mail:<bernardo.sfs27@gmail.com> e <andresalima625@gmail.com>
[4]Graduando em Ciência da Computação - UFT. e-mail:< pauloatavilapa@gmail.com>
[5]Professor Doutor em Engenharia Elétrica - IFTO. e-mail: <balduino@ifto.edu.br>

Resumo: O estudo realizado neste trabalho tem trata do desenvolvimento de ferramenta computacional para resolver problemas de otimização combinatória envolvendo estatística, implementada nas linguagens de programação C, C++ e/ou Java. A classificação de demandas por afinidades é a linha básica das soluções pesquisadas e pode ser exemplificada com o problema de, em uma turma onde cada indivíduo manifesta certo grau de interesse em trabalhar com outros colegas, criar grupos de trabalho que respeite os interesses individuais e aponte a melhor configuração do agrupamento pretendido. Situações semelhantes são observadas na distribuição de energia elétrica entre as fontes geradoras e os grandes centros consumidores, no transporte da produção agropecuária até os centros de abastecimentos urbanos, nas especificações de tecnologias para o roteamento de sinais e canais de telecomunicações e em grande número de problemas onde se busca uma solução para a organização de um grupo de elementos com características diferenciadas. Neste trabalho, é apresentada a metodologia utilizada pelos pesquisadores para elaborar algoritmos que geram os agrupamentos possíveis para uma determinada quantidade de elementos. Aplicando os conhecimentos de probabilidade e estatística foram formuladas equações que auxiliam na implementação de algoritmos que geram listas com todos os agrupamentos possíveis para n elementos agrupados p a p. Os resultados obtidos indicam o caminho para chegar à ferramenta computacional capaz de separar elementos classificando-os por afinidade.

Palavras–chave: algoritmo, agrupamentos, classificação, combinação, probabilidade.

1 INTRODUÇÃO

No universo acadêmico observa-se com frequência o desenvolvimento de trabalhos em grupo. Porém pouca atenção é dada à forma que os grupos são elaborados e as informações que este processo pode revelar aos educadores. Quando o professor determina a composição dos grupos, por mais que o educador tenha uma noção dos relacionamentos que se fazem na sala escolar, é natural que nem todos os estudantes possam ter suas vontades atendidas. Atribuindo aos estudantes a responsabilidade de se auto organizarem em grupos, expõem-se os estudantes que possuem um comportamento tímido, e que algumas vezes não são aceitos por parte dos colegas, a situações constrangedoras.

A ferramenta que este projeto se propõe a desenvolver aponta a melhor solução para situações desta natureza, com alta relevância para o processo de ensino/aprendizagem e permitirá o estudo futuro das informações sobre as afinidades entre os alunos de uma mesma turma e o impacto no sucesso da vida acadêmica.

Tais soluções são ilustradas no estudo de estatística e análise combinatória por meio de exemplos clássicos de aplicações de similar complexidade, certificando que os resultados desejados são viáveis de serem alcançados com os esforços da equipe de pesquisadores composta.

ISSN 2179-5649

Objetivo geral

O objetivo geral deste projeto é desenvolver conhecimentos que permitam produzir rotinas para solucionar os problemas de Classificação de Demandas por Afinidades, aplicando técnicas de otimização combinatória.

Objetivos específicos

Promover a melhoria do processo de ensino/aprendizagem por meio de ferramentas computacionais inovadoras;

Desenvolver as habilidades dos pesquisadores na programação computacional em linguagens de programação C, C++ e/ou Java;

Explorar novas soluções que exemplificam a aplicação de técnicas de otimização combinatória;

Oferecer ao corpo docente ferramenta para o aprimoramento da elaboração de grupos de pessoas com atividades em comum;

Sustentar o desenvolvimento de pesquisas futuras que tratam dos relacionamentos interpessoais do corpo discente e que identifiquem previamente os distúrbios comportamentais.

2 REFERENCIAL TEÓRICO/ESTADO DA ARTE

O raciocínio combinatório é importante para o raciocínio lógico matemático e para a relação com outras áreas do conhecimento, como na lógica de programação, uma vez que sua utilidade vai além da teoria. Segundo Guirado e Cardoso (2007), o raciocínio combinatório pode ser aplicado em diversas áreas como no cálculo das probabilidades, de programação linear, de estatística, de teoria da informação, de lógica, etc. De acordo com Pessoa e Borba (2010) esse conhecimento se aproxima ainda mais da realidade das pessoas, sendo que problemas de combinatória são explorados desde cedo, pois expectativas de um acontecimento, regras de um jogo, escolha de vestimentas, escolha de rotas, são ricas situações que estimulam o desenvolvimento e exercitam o raciocínio lógico.

A análise combinatória, como Merayo (2001) defende, é a técnica de saber quantos objetos há em um conjunto sem realmente ter que contá-los, porque essa técnica não necessita listar ou enumerar todos os elementos que formam o conjunto. Dessa maneira, a contagem na Análise Combinatória não é vista meramente como enumeração direta de elementos, mas como determinação de possibilidades sem necessariamente levantar todos os casos possíveis.

A representação dos resultados obtidos pode ser feita utilizando diferentes linguagens - verbal, matemática, gráfica, de programação - com o objetivo de produzir e expressar ideias, interpretando diferentes situações. Neste trabalho vamos utilizar da linguagem de programação para expressar nossos resultados e conciliar a estatística, análise combinatória e a programação.

Para se chegar a um resultado é possível que se obtenha várias, ou possivelmente infinitas soluções. Assim surge o problema otimização que visa obter um projeto ótimo, maximizando ou minimizando as variáveis desejadas.

A resolução de um problema de otimização normalmente precisa passar por duas fases: a primeira é transformar o problema em um modelo e, posteriormente, implementar um algoritmo para resolver o modelo.

Neste trabalho são concentrados os esforços na primeira fase de modelamento, com elaboração de algoritmos para os problemas de menor complexidade, considerando que

> Nem sempre é possível encontrar a melhor solução de um problema de otimização em tempo razoável por meio de algoritmos exatos. Nesses casos, uma solução relativamente boa pode já ser suficiente para a aplicação que se tem em mãos. Os Métodos Heurísticos são algoritmos que não garantem encontrar a solução ótima de um problema, mas são capazes de retornar uma solução de qualidade em um tempo adequado para as necessidades da aplicação (CONSTANTINO).

3 METODOLOGIA/MATERIAIS E MÉTODOS

As etapas ora apresentadas são de cunho descritivo com objetivo de expor a sistemática utilizada nas atividades realizadas.

A partir de reuniões realizadas periodicamente entre pesquisadores e orientador definem-se os estudos necessários para dar andamento ao projeto, iniciando-se pela elaboração de listas sequenciais de reduzida quantidade de elementos, que são as combinações de grupos possíveis para N elementos agrupados P a P.

Com base na lista montada e verificada, os pesquisadores começam a buscar um padrão de comportamento, visando obter uma equação que define a quantidade de combinações e de agrupamentos. A partir desta equação é desenvolvido um fluxograma para que a lista seja gerada por um algoritmo. E por fim o fluxograma é implementado em linguagem de programação, o programa gera novas listas que são comparadas com as que foram elaboradas anteriormente, bem como se o número total condiz com o resultado da equação.

Caso duas lógicas sejam apresentadas, as duas são implementadas e é comparado o tempo de execução, a mais rápida é a que será utilizada. Confirmada a validez do algoritmo é realizada uma nova reunião para discutir o próximo passo do projeto.

Aplicando o método já apresentado foram realizados dois estudos, o primeiro com o objetivo de mostrar o número de agrupamentos possíveis quando n pessoas forem divididas em dois grupos e o segundo quando divididas em três grupos.

3.1 N pessoas divididas em dois grupos

Através dos conhecimentos de probabilidade e estatística, foi estudada a fórmula de combinação (fórmula 1):

$$C_{n,p} = \frac{n!}{p!(n-p)!} \quad (1)$$

Onde,
n é a quantidade de elementos de um conjunto, e
p é a quantidade de elementos que irão formar os agrupamentos.

Na combinação, a ordem dos elementos não é considerada na formação dos subconjuntos, ou seja, o subconjunto {A, B} e {B, A} são iguais. Sendo assim, seguindo a fórmula de combinações 8 pessoas divididas em dois grupos, resultaria em 70 possibilidades:

$$p = \frac{n}{2} = 4$$

$$C_{8,4} = \frac{8!}{4!(8-4)!} = 70$$

Mas, com este resultado, estaríamos desconsiderando a ordem, e teríamos grupos repetidos, como:

Grupo 1: 1; 2; 3; 4. 5; 6; 7; 8.
Grupo 2: 3; 2; 4; 1. 8; 5; 7; 6.

G1 e G2 possuem os mesmos elementos, mas em ordens diferentes, que para o estudo do projeto, correspondem ao mesmo grupo.

Para descobrir então o número de combinações sem as repetições, foram determinadas, manualmente, combinações possíveis de n = 4 e p = 2; n = 6 e p = 3; n = 8 e p = 4; n = 10 e p = 5; e através dos resultados Concluiu-se que para dividir n pessoas com $p = \frac{n}{2}$ sem que haja repetições, aplica-se a expressão:

$$R_{n,p} = \frac{\frac{n!}{p!(n-p)!}}{2} \quad (2)$$

Onde R (fórmula 2) aponta o número total de combinações possíveis.

Para gerar as combinações possíveis foi criado um programa em linguagem C. No Início do programa é determinada a quantidade elementos que são divididos (G), e divide-se essa quantidade total por dois e é obtida a quantidade de pessoas por grupo (P). São criados três vetores: ordem, principal e base.

O vetor principal registra onde as alterações serão feitas, onde os números serão trocados, e logo após exibidos na tela. O vetor ordem armazena, durante todo o tempo de execução do programa, a ordem crescente dos elementos do programa. O vetor base é alterado periodicamente durante o tempo de execução do programa, porém suas alterações não são tão frequentes quanto o vetor principal.

Após a criação dos vetores é iniciado o preenchimento dos três vetores na ordem crescente de 1 a G. Logo após, são aplicados dois processos aos vetores, os processos são chamados de conta lista e monta lista.

O vetor principal é dividido em duas partes: os dois grupos a serem obtidos. Como só foi criado um vetor para os dois grupos, considera-se da primeira posição até a metade como grupo 1 e da metade até o final como grupo 2.

O primeiro processo pelo qual o vetor principal passa é o conta lista, onde a última posição do grupo 1 é comparada com todas as posições do grupo 2. Caso o número da posição do grupo 2 seja maior que o número na última posição do grupo 1 os números são trocados de posição, através de uma função usando ponteiros. Toda vez que uma troca é realizada é exibida na tela a nova combinação. Como o vetor principal é alterado, o vetor base que é usado para comparar o grupo 1 com o grupo 2 é alterado no final do processo monta lista.

Após o processo anterior (conta lista) ser aplicado ao vetor, o programa aplica o segundo processo que é chamado de monta lista. Essa parte do programa serve para reorganizar o vetor depois que foram feitas todas as trocas possíveis na última posição do grupo 1, a partir da penúltima posição do grupo 1 até a primeira. Em seguida, é verificado se a posição já alcançou seu número limite (soma entre o índice I da posição e P, a quantidades de membros por grupo). Caso não tenha atingido seu número limite, a lista é reorganizada a partir desta posição. Após essa reorganização ser feita o vetor principal é exibido na tela e o vetor base recebe os mesmos valores que estão no vetor principal.

Ao final dos processos conta lista e monta lista é verificado se a segunda posição do vetor já atingiu seu limite, caso não tenha atingido seu limite o conta lista e o monta lista são executados novamente, até que essa condição seja satisfeita. É usada a segunda posição como parâmetro, pois quando a primeira é alterada obtemos combinações repetidas. Quando o programa finaliza todas as etapas são mostradas na tela as combinações geradas.

3.2 N pessoas divididas em três grupos

Inicialmente foi elaborada uma lista teste com 6 elementos agrupados 2 a 2, e foi feita uma análise de forma separada de cada um dos três agrupamentos. Para definir o primeiro agrupamento, fixou-se o primeiro elemento, para que evitasse repetições, e assim aplica-se a fórmula de combinações com n-1 e p-1, já que o primeiro elemento está definido e este não compõe a parte variante, e este resultado será chamado de R1 (fórmula 3).

$$R1_{n,p} = \frac{(n-1)!}{(p-1)!((n-1)-(p-1))!} \quad (3)$$

Tendo o primeiro agrupamento definido, foi aplicada a fórmula (1), substituindo-se n por $n-p$, pois os elementos definidos no agrupamento 1 não irão variar nos agrupamentos seguintes. Este resultado foi chamado de R2 (fórmula 4).

$$R2_{n,p} = \frac{\frac{(n-p)!}{p!((n-p)-p)!}}{2} \quad (4)$$

Por fim, determinamos o resultado final multiplicando R1 e R2 (fórmula 5):

$$Comb = R1 * R2 \quad (5)$$

Onde Comb é o número de combinações possíveis.

A rotina foi implementada em linguagem C, em processo semelhante ao especificado anteriormente.

A mesma sistemática será utilizada em estudos futuros com o objetivo de formular uma equação capaz de determinar a quantidade de agrupamentos para qualquer n elementos agrupados p a p.

4 RESULTADOS E DISCUSSÕES

A lógica para mostrar o número de combinações possíveis quando n pessoas forem divididas em dois e três grupos foram implementadas em linguagem de programação. Os programas feitos conseguem determinar a quantidade de grupos que podem ser feitos com P=G/2 e P=G/3, onde G é a quantidade total de elementos e P é a quantidade de elementos por grupo.

Na Figura 1 e Figura 2 pode-se observar os agrupamentos gerados pelo algoritmo implementado em linguagem de programação C++. Para agrupamentos onde G é um valor pequeno, como foi utilizado nos exemplos mostrados nas Figuras 1 e 2, o tempo de execução do programa se mostrou bastante satisfatório, porém, para valores de G maiores, como por exemplo, G=15, o programa leva um tempo maior para conseguir gerar os agrupamentos.

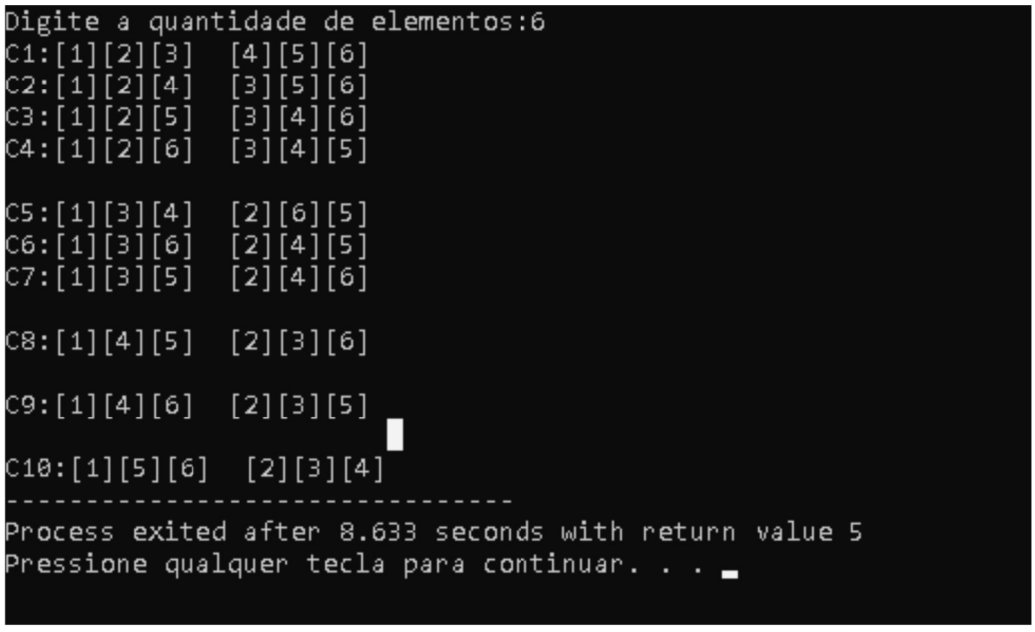

Figura 1- Agrupamentos para G=6 e P=G/2 gerados pelo programa implementado em C++.

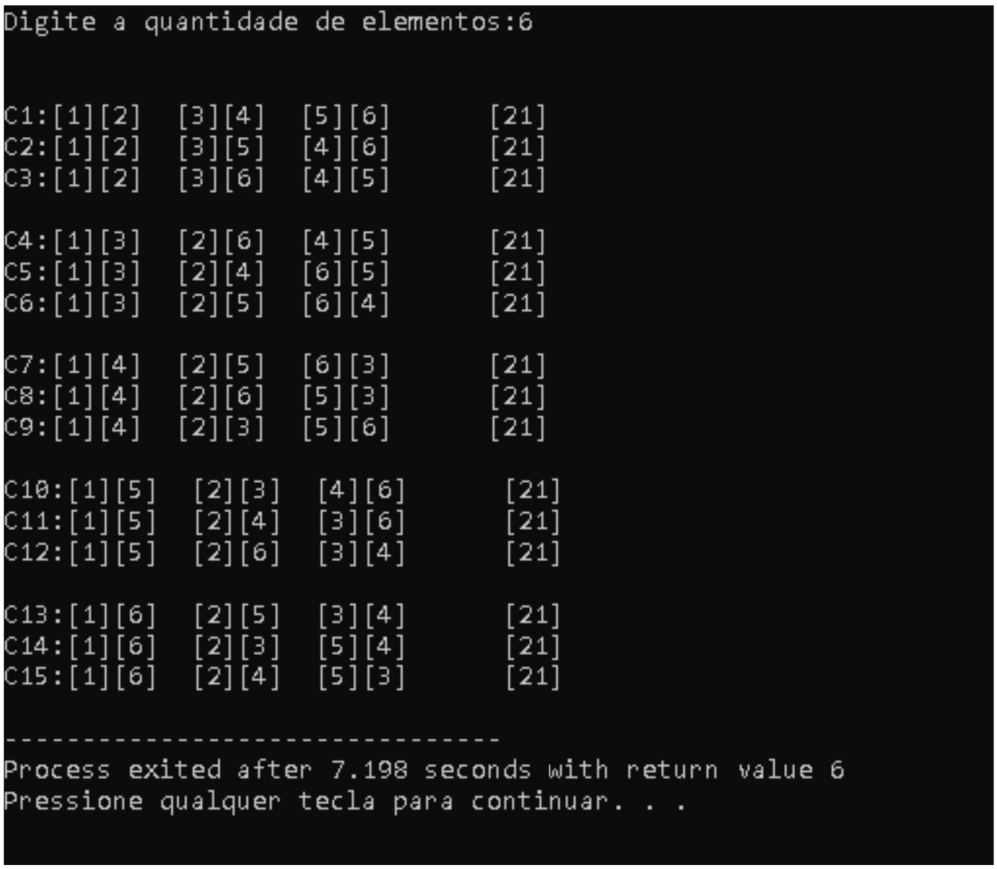

Figura 2-Agrupamentos para G=6 e P=G/3 gerados pelo programa implementado em C++.

Observando os resultados esperados, conclui-se que o trabalho apresentou resultados satisfatórios. Foi possível verificar os aspectos que precisam ser aprimoradas para que os estudos futuros sejam mais abrangentes e que consigam separar os elementos em uma maior quantidade de grupos.

Deve ser dada uma atenção especial para a otimização, uma vez que quando G é um número grande o tempo de execução para os programas implementados é consideravelmente maior para elaborar os agrupamentos.

5 CONCLUSÃO OU CONSIDERAÇÕES FINAIS

Considera-se que o estudo desenvolvido contribuiu para o desenvolvimento e aprendizagem, assim como aponta, também, a relevância de uma ferramenta computacional capaz de separar elementos classificando-os por afinidade. A exploração deste estudo favoreceu a compreensão de que o uso da probabilidade e estatística possibilita a solução de problemas cotidianos, e pode ser aplicada em diversas áreas de estudo, como no caso do projeto em questão que propõe um algoritmo para oferecer ao corpo docente uma ferramenta para tratar dos relacionamentos interpessoais do corpo discente e que identifiquem previamente os distúrbios comportamentais.

Para aprimorar os estudos são recomendados os estudos de novas aplicações que exemplificam o uso das técnicas de otimização combinatória, reduzindo o tempo de execução e assim outros trabalhos poderão ser desenvolvidos aplicando esse mesmo sistema às novas possibilidades que devem ser desenvolvidas.

Por fim, é possível afirmar que os resultados preliminares obtidos foram satisfatórios e próximos ao previsto.

REFERÊNCIAS

AGUILAR, Luis Joyanes. **Programação em C++: algoritmos, estruturas de dados e objetos**. São Paulo: McGraw-Hill, 2008.

BARNES, David J. KÖLLING, Michael. **Programação orientada a objetos com Java**. 4. ed. São Paulo: Pearson, 2009.

CONSTANTINO, Ademir aparecido. **"Projeto e Análise de Algoritmos" + "Estrutura de Dados" + "Matemática" = "Otimização Combinatória"**. Disponível em http://www.din.uem.br/~ademir/Otimizacao.html <Acesso em 05/05/2017>

CRESPO, Antônio A. **Estatística Fácil**. 17ª ed. – São Paulo: Saraiva, 2002.

FONSECA, Jairo S. e Gilberto de Andrade M. **Curso de Estatística**. 6ª ed. – São Paulo: Atlas, 1996.

GELUPPO, Fabio. MATHEUS, Vanclei. SANTOS, Wallace. **Desenvolvendo com C#**. Porto Alegre: Bookman, 2004.

MEYERS, Scott**. C++ Eficaz: 55 maneiras de aprimorar seus programas e projetos**. 3. ed. Porto Alegre: Bookman, 2011.

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C**. São Paulo: Pearson, 2008. .

Monalisa Cardoso Silva, Cristiane Azevedo dos Santos Pessoa. **A Combinatória: estado da arte em anais de eventos científicos nacionais e internacionais ocorridos no Brasil de 2009 a 2013**. Educ. Matem. Pesq., São Paulo, v.17, n.4, pp.670-693, 2015.

SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Rio de Janeiro: Elsevier, 2003.

STROUSTRUP, Bjarne. **A linguagem de programação C++**. 3. ed. Porto Alegre: Bookman, 2000.