JONATHAN RIBEIRO DOS SANTOS
SANDRO AUGUSTO DE OLIVEIRA

ALGORITMOS GENÉTICOS APLICADOS À LINHA DE PRODUÇÃO DE CALÇAS

UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE – MG
2015

JONATHAN RIBEIRO DOS SANTOS
SANDRO AUGUSTO DE OLIVEIRA

ALGORITMOS GENÉTICOS APLICADOS À LINHA DE PRODUÇÃO DE CALÇAS

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação da Universidade do Vale do Sapucaí como requisito parcial para obtenção do título de bacharel de Sistemas de Informação.

Orientador: Prof. Me. Roberto Ribeiro Rocha

UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE – MG
2015

Santos, Jonathan Ribeiro dos; Oliveira, Sandro Augusto de

Algoritmos genéticos aplicados à linha de produção de calças / Jonathan Ribeiro dos Santos, Sandro Augusto de Oliveira – Pouso Alegre – MG: Univás, 2015.
68 f. : il.

Trabalho de Conclusão de Curso (graduação) – Universidade do Vale do Sapucaí, Univás, Sistemas de Informação.
Orientador: Prof. Me. Roberto Ribeiro Rocha

1. Algoritmos genéticos. 2. evolução. 3. espécies.

JONATHAN RIBEIRO DOS SANTOS
SANDRO AUGUSTO DE OLIVEIRA

ALGORITMOS GENÉTICOS APLICADOS À LINHA DE PRODUÇÃO DE CALÇAS

Trabalho de conclusão de curso defendido e aprovado em 01/01/2014 pela banca examinadora constituída pelos professores:

Prof. Me. Roberto Ribeiro Rocha
Orientador

Prof^{a}. ME. Nome da professora
Avaliadora

Prof. MSc. Nome do professor
Avaliador

De Jonathan Ribeiro dos Santos.

Dedico este trabalho . . .

De Sandro Augusto de Oliveira.

Dedico este trabalho . . .

AGRADECIMENTOS

De Jonathan Ribeiro dos Santos

Agradeço . . .

De Sandro Augusto de Oliveira

Agradeço . . .

“Complicar é simples,
simplificar que é complicado.
(Paulo Sérgio dos Santos)

SANTOS, Jonathan Ribeiro dos; Oliveira, Sandro Augusto de. **Algoritmos genéticos aplicados à linha de produção de calças**. 2015. Monografia – Curso de SISTEMAS DE INFORMAÇÃO, Universidade do Vale do Sapucaí, Pouso Alegre – MG, 2015.

RESUMO

Os algoritmos genéticos, inspirados na genética e na teoria da evolução das espécies de Charles Darwin, são algoritmos probabilísticos que, aleatoriamente, promovem uma busca paralela pela melhor solução de um problema e são desenvolvidos baseados no princípio de sobrevivência dos mais aptos, na reprodução e na mutação dos indivíduos, de forma a fazer com que soluções evoluam e se tornem cada vez melhores. A presente pesquisa apresenta uma visão geral sobre o conceito de algoritmos genéticos, demonstrando seus fundamentos, operadores e suas configurações, realizando um comparativo com os conceitos semelhantes na natureza. Para ilustrar a utilização desta técnica, foi desenvolvida uma aplicação, em plataforma WEB, juntamente com um algoritmo genético, que visa encontrar soluções otimizadas para distribuição das atividades de uma sistema de produção de calças em um modo de se produzir que permite que os empregados trabalhem em suas casas e que cada parte da calça é confeccionada separadamente. O software recebe, como entrada, dados de tempo e preço de produção por peça de cada empregado e, após a execução do algoritmo genético sobre estes dados, apresenta como saída uma boa forma de distribuição das atividades que permita que a produção seja realizada com o menor tempo e tenha o menor custo possível dentro do prazo de entrega. Esta pesquisa é do tipo aplicada, pois não visa modificar os processos de produção e sim apresentar formas de otimização desse.

Palavras-chave: Algoritmos genéticos. evolução. espécies.

SANTOS, Jonathan Ribeiro dos; Oliveira, Sandro Augusto de. **Algoritmos genéticos aplicados à linha de produção de calças**. 2015. Monografia – Curso de SISTEMAS DE INFORMAÇÃO, Universidade do Vale do Sapucaí, Pouso Alegre – MG, 2015.

ABSTRACT

The Genetic algorithms, inspired in genetic and in the Charles Darwin's theory of evolution of species, are probabilist algorithms which is used to, randomly, make a parallel search for the best solution for a given problem. It is implemented based on principle of survival of the fittest, on the reproduction and on mutation of individuals and so it makes possible the evolution of the solutions which can make them even better. This research presents a general view of genetic algorithms concept, it demonstrate its fundamentals, operators and configurations and also makes a comparison with similar concepts in nature. To illustrate how this technique works it was developed an WEB based application with a genetic algorithm which was developed to find optimized solutions for tasks distribution in a production system of pants in which the employees work in their own houses and each part of the pants is produced separately. The software receives as input the time and the price of production per part of each employee and, after execution of the genetic algorithm over this data, it shows one good form to distribute the tasks in order to make the production in the shortest time and in the shortest cost within the planned deadline. This kind of research is applied because it will not modify the production processes, it will just optimize them.

Key words: Genetic Algorithms. evolution. species.

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

ABNT	Associação Brasileira de Normas Técnicas
API	*Application Programming Interface*
GNU	*Gnu is Not Unix*
GPL	*General Public License*
MVC	*Model – View – Controller*

SUMÁRIO

INTRODUÇÃO

Sabe-se que atualmente existem diversos softwares disponíveis no mercado. Programas que vão desde aqueles desenvolvidos sob medida até softwares genéricos popularmente chamados de "softwares de prateleira". Porém no cenário corporativo, levando em consideração que empresas buscam a cada dia se tornarem mais competitivas, é necessário que um sistema ofereça suporte para que estas possam se tornar mais eficientes, como por exemplo, auxiliar na busca pela diminuição dos custos operacionais para que assim seja possível alcançar os melhores preços de venda. (LAUDON; LAUDON, 2009)

Neste contexto, a ideia de agregar características semelhantes à inteligência humana aos programas se torna uma alternativa interessante, pois segundo Laudon e Laudon (2009, p.329),

> Técnicas inteligentes ajudam os tomadores de decisão capturando o conhecimento coletivo e individual, descobrindo padrões e comportamentos em grande quantidade de dados e gerando soluções para problemas amplos e complexos demais para serem resolvidos por seres humanos.

O conceito por trás deste pensamento é denominado Inteligência Artificial - IA[1] - e é definido por Luger e Stubblefield (1993), como uma área da ciência da computação que abrange a automatização da inteligência.

Para Luque e Silva (2010, p.44), "a IA é inspirada em processos naturais e está relacionada à reprodução de capacidades normalmente associadas à inteligência humana, como aprendizagem, adaptação, o raciocínio, entre outras". Ainda segundo os mesmos autores, várias abordagens surgiram ao longo da história tais como a abordagem conexionista, inspirada nos neurônios biológicos, a simbolista, baseada na inferência humana e a evolutiva, fundamentada na teoria de evolução das espécies.

Na busca por otimizar seu processo de produção, uma empresa da região, que fabrica calças e aloca costureiras que trabalham em suas casas, deseja saber qual a melhor forma de distribuir o trabalho para que um determinado lote de seu produto seja produzido no menor tempo possível e com o menor custo. Para a resolução deste problema, pode se fazer uso de IA através de um dos ramos da abordagem evolutiva denominado Algoritmos Genéticos - AGs[2] - pois, como afirma Fernandes (2003), os AGs resolvem problemas de otimização através de um processo que oferece como saída a melhor solução dentro de várias possíveis formas de se resolver um problema.

---

1 O termo Inteligência Artificial será referenciado pela sigla IA a partir deste ponto do trabalho.

2 O termo Algoritmos Genéticos será referenciado pela sigla AGs a partir deste ponto do trabalho.

O presente trabalho têm por objetivo geral, desenvolver uma aplicação utilizando técnicas de inteligência artificial, capazes de realizar a alocação de empregados em uma linha de produção de forma otimizada. Para a realização do mesmo foram colocados os seguintes objetivos específicos: a) Demonstrar o uso de algoritmos genéticos; b) Projetar uma aplicação em plataforma WEB que distribua as atividades de uma linha de produção de calças de forma inteligente, a fim de se obter o menor custo e o menor tempo de produção.

Para Linden (2012), AGs é definido como uma técnica de otimização e busca que se baseia na teoria do processo de evolução e seleção natural, proposto por Charles Darwin em seu livro A Origem das Espécies, que afirma que indivíduos com melhor capacidade de adaptação ao seu ambiente possuem maior chance de sobreviver e gerar descendentes.

Segundo Fernandes (2003), o termo foi proposto por Holland, em 1975, e por imitação à teoria da evolução, é representado por uma população de indivíduos que representam soluções para um determinado problema e então tais soluções podem evoluir até se chegar a uma solução ótima.

Vários trabalhos foram desenvolvidos utilizando a robustez de AGs, dentre eles o trabalho de Santos et al. (2007), que faz a seleção de atributos usando AGs para classificação de regiões, o trabalho de Silva (2001), que descreve a otimização de estruturas de concreto armado utilizando AGs e o trabalho de Freitas et al. (2007) que descreve uma ferramenta baseada em AGs para a geração de tabela de horário escolar.

O sistema desenvolvido neste trabalho, assim como nos outros citados, tem o mesmo conceito, o de otimizar processos. No caso da fábrica de calças, tal otimização irá promover a diminuição do custo de produção das calças confeccionadas, permitindo assim com que estas possam ser vendidas também por um preço melhor, beneficiando assim, os consumidores da região. Além disso, a ideia de otimizar procedimentos, muitas vezes, também contribui para a preservação de recursos naturais devido ao fato de que processos otimizados podem significar economia de energia e diminuição de emissão de gases. No caso da empresa de calças, se o gestor ter sempre em mãos a solução mais otimizada, irá ter um menor tempo de transporte de materiais, o que irá reduzir a emissão de gases dos veículos utilizados no transporte de materiais e peças entre as costureiras.

No âmbito acadêmico, o trabalho agregará à base de conhecimento da Univás um material que faça referência a tecnologias e conceitos de inteligência artificial, que hoje estão presentes em diversos sistemas críticos de apoio a decisão e otimização de processos nas empresas.

2 QUADRO TEÓRICO

Neste capítulo serão listados os conceitos e as tecnologias que serão utilizados no desenvolvimento da proposta de trabalho apontada na seção objetivos. Para tal, serão discutidos a definição, o histórico e as aplicabilidades de cada um deles, tomando por base autores fundantes e seus comentaristas. É importante ressaltar que o texto desta seção, quando descreve a teoria da evolução das espécies, não tem como objetivo levantar questões sobre a origem dos seres vivos.

2.1 Algoritmos Genéticos

Nesta seção será descrito como surgiu, conceitos e algumas características dos Algoritmos Genéticos, tema principal deste trabalho e fundamental para o desenvolvimento da aplicação.

2.1.1 Fundamentos

Para Melanie (1999), desde o começo da era computacional, cientistas pioneiros, tais como Alan Turing, John von Neumann, Norbert Wiener e outros, tinham o objetivo de dotar os computadores de inteligência de maneira que eles pudessem tomar decisões, se adaptar a determinadas situações e até mesmo ter a capacidade de aprender. Com esta motivação, estes cientistas se interessaram por outras áreas, além da eletrônica, como a biologia e a psicologia, e começaram então a realizar pesquisas para simular os sistemas naturais no mundo computacional a fim de alcançarem suas metas.

Vários conceitos computacionais baseados na natureza surgiram então ao longo do tempo, dentre eles, a computação evolucionária inspirada na teoria da evolução natural, da qual o exemplo mais proeminente são os AGs que foram introduzidos por Jhon Holland, seu aluno David Goldberg e outros estudantes da universidade de Michigan. Goldberg (1989) define os AGs como métodos de busca baseados na genética e no mecanismo de seleção natural que permitem a possibilidade de obter robustez e eficácia na tarefa de encontrar uma boa solução para um problema em um espaço de busca complexo, em um tempo aceitável.

Segundo Linden (2012), a teoria da evolução foi proposta pelo naturalista inglês Charles Darwin por volta de 1850, quando este, em uma viagem de navio visitou vários lugares e por ser uma pessoa com uma grande habilidade de observação, percebeu que indivíduos de uma mesma espécie vivendo em lugares diferentes possuíam também características distintas, ou seja, cada indivíduo possuía atributos específicos que lhe permitia uma melhor adaptação em seu ecossistema.

O autor afirma então que, com base nesta observação, Darwin propôs que existe um processo de seleção natural, afirmando que, como os recursos na natureza, tais como água e comida, são limitados, os indivíduos competem entre si e aqueles que não possuem atributos necessários à adaptação ao seu ambiente tendem a ter uma probabilidade menor de reprodução e irão ser extintos ao longo do tempo e, por outro lado, aqueles com características que os permitem obter vantagens competitivas no meio onde vivem, acabam tendo mais chances de sobreviver e gerar indivíduos ainda mais adaptados.

A teoria ressalta porém que o processo não tem o objetivo de maximizar algumas características das espécies, pois os novos indivíduos possuem atributos que são resultados da mesclagem das características dos reprodutores, o que faz com que os filhos não sejam exatamente iguais aos pais, podendo assim ser superiores, uma vez que, estes herdem as qualidades de seus pais ou inferiores se os descendentes herdarem as partes ruins de seus reprodutores (LINDEN, 2012).

Para entender a relação entre AGs e a evolução natural é necessário conhecer as principais terminologias biológicas, sendo importante ressaltar porém que, de acordo com Melanie (1999), apesar da analogia a certos termos da biologia, a forma com que os AGs são implementados é relativamente simples se comparado ao funcionamento biológico real.

Melanie (1999) afirma que todos os seres vivos são compostos de células e estas possuem um ou mais cromossomos que, basicamente, são manuais de instruções que definem as características do organismo. O cromossomo é formado por um conjunto de genes que, em grupo ou individualmente, são responsáveis por um determinado atributo do indivíduo como por exemplo, a cor do cabelo, a altura, etc. Cada gene possui uma localização dentro do cromossomo denominada *locus* e, por fim, o conjunto de todos os cromossomos dentro da célula é definido como genoma.

Considerando isto, Linden (2012, p.33) afirma que,

> Um conjunto específico de genes no genoma é chamado de genótipo. O genótipo é a base do fenótipo, que é a expressão das características físicas e mentais codificadas pelos genes e modificadas pelo ambiente, tais como cor dos olhos, inteligência etc. Daí, podemos concluir: nosso DNA codifica toda a infor-

> mação necessária para nos descrever, mas esta informação está sob controle de uma grande rede de regulação gênica que, associada às condições ambientais, gera as proteínas na quantidade certa, que farão de nós tudo aquilo que efetivamente somos.

Uma vez descrita a complexidade dos organismos é necessário discorrer, de forma básica, sobre o processo de reprodução responsável pela transmissão da informação genética de geração para geração.

Linden (2012) afirma que existem dois tipos de reprodução, a assexuada, em que não é necessário a presença de um parceiro e a sexuada que exige a presença de dois organismos. Os AGs simulam a reprodução sexuada em que cada um dos organismos envolvidos oferece um material genético denominado gametas. Os gametas são formados por um processo denominado *crossing-over* ou *crossover* que tem início com a divisão de cada cromossomo em duas parte as quais irão se cruzar uma com a outra para formar dois novos cromossomos, que receberão um pedaço de cada uma das partes envolvidas no cruzamento.

Ainda segundo o autor, o resultado deste processo serão, então, quatro cromossomos potencialmente diferentes que irão compor os gametas e farão parte dos novos indivíduos. Neste processo, podem ocorrer mutações que são resultados de alguns erros ou da influência de algum fator externo, como a radiação por exemplo. Estas mutações são pequenas mudanças nos genes dos indivíduos, podendo estas ser boas, ruins ou neutras.

E assim a informação genética é passada dos pais para os filhos, e como os componentes dos cromossomos definem as características do organismo, os filhos herdarão características dos pais, porém serão ligeiramente diferente deles, como foi descrito anteriormente, o que permite que os novos indivíduos herdem características melhores ou piores que seus progenitores, porém, se os pais possuem características positivas, a probabilidade de gerarem filhos ainda melhores são maiores (LINDEN, 2012).

2.1.2 Características dos Algoritmos Genéticos

De acordo com Linden (2012), a analogia dos AGs com os processos biológicos se dá por meio da representação de cada termo descrito anteriormente em um modelo computacional voltado a encontrar soluções para um determinado problema em um processo aleatório. O fluxo de execução deste processo inicia-se com a criação aleatória de uma população inicial. Uma população contém um conjunto de indivíduos sendo que cada indivíduo representa uma possível solução para o problema.

O autor afirma ainda que um indivíduo é formado por cromossomos que guardam as características da solução, ou seja, uma forma de resolver o problema.

Segundo Melanie (1999), a execução de um algoritmo genético é basicamente realizada conforme os itens a seguir:

1. Definição da população inicial;
2. Avaliação e classificação dos indivíduos da população;
3. Seleção de acordo com a qualidade;
4. Cruzamento para geração de novos descendentes;
5. Mutação aleatória dos novos indivíduos;
6. Repetição do processo de seleção, cruzamento e mutação até se formar uma nova população;
7. Avaliação e classificação dos indivíduos da nova população;
8. A nova população substitui a anterior e o processo continua a partir do intem 2 até que o número de populações criadas atinja um limite, que é definido previamente, ou até atingir outra condição definida pelo programador.

A Figura 1 ilustra os passos do algoritmo.

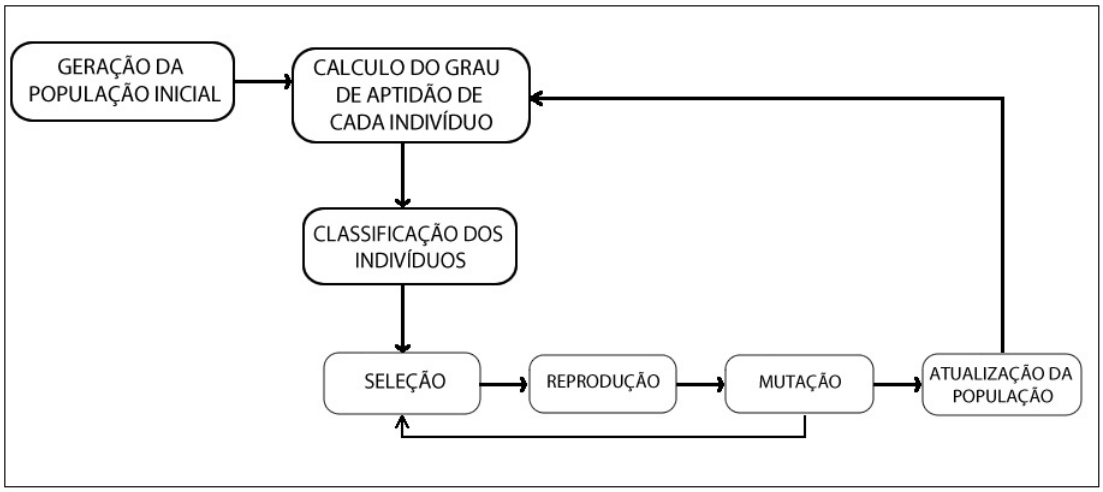

Figura 1 – Demonstração da execução de um AG **Fonte:** Desenvolvido pelos autores.

Como demostrado acima existem dois `loops`, o `loop` que acontece até que uma nova população seja formada e outro responsável por iniciar a criação de uma nova população até que o número de gerações seja atingido.

2.1.2.1 População Inicial

De acordo com Melanie (1999), a população inicial é o conjunto dos primeiros indivíduos candidatos à resolução do problema. Estes indivíduos devem ser criados aleatoriamente, seguindo a lógica definida pelo programador com base no contexto do problema a ser resolvido. Um exemplo seria a população inicial do algoritmo desenvolvido neste trabalho, em que a população inicial será formada por possíveis formas de se produzir um determinado lote de calças, ou seja, será formada uma população de possíveis formas de dividir o trabalho entre as costureiras.

2.1.2.2 Função de avaliação

Segundo Linden (2012), após a criação da população inicial, esta é então avaliada através de uma função de avaliação que mede a qualidade de cada uma de suas soluções e é realizada então uma classificação que ordena as soluções das melhores para as piores, para então iniciar a formação de uma nova população, a nova população pode conter, já inicialmente, os dois melhores indivíduos existentes na população inicial, este mecanismo é denominado elitismo e pode ser utilizado ou não.

Ainda segundo o autor, a função de avaliação ou função de aptidão penaliza as soluções inviáveis para a solução do problema, ou seja, ao verificar que uma certa solução não satisfaz o grau de aptidão necessário para o problema proposto, esta solução é descartada e, assim, só sobrevivem as soluções com mais chance de resolver o problema e por isso torna-se o componente mais importante de qualquer algoritmo genético.

Devido à generalidade encontrada nos AGs, a função de avaliação torna-se em muitos casos a única ligação verdadeira entre o programa e o problema real pois ela irá analisar os cromossomos de cada indivíduo, convertendo-os assim, em uma proposta de solução para o problema (LINDEN, 2012).

2.1.2.3 Seleção

Segundo Linden (2012), o próximo passo para a formação da nova população é o cruzamento, que se inicia com a seleção de dois indivíduos, que é realizada de acordo com a

qualidade destes, porém, como é fundamental que esta escolha não despreze completamente os indivíduos com uma qualidade muito baixa, a seleção é feita de forma probabilística, ou seja, indivíduos com boa qualidade possuem mais chances de serem selecionados e, por outro lado, indivíduos com menor nota de avaliação terão menos chance se reproduzirem.

O autor ressalta que a razão de o mecanismo de seleção não escolher apenas as melhores soluções é devido ao fato de que aquelas com menor grau de avaliação, também serem importantes para que se tenha uma maior diversidade de características na população envolvida para a solução do problema, possibilitando assim que esta possa evoluir de forma satisfatória, pois se as novas populações forem constituídas somente das melhores soluções, elas serão compostas de indivíduos cada vez mais semelhantes impedindo assim que novas soluções ainda melhores sejam concebidas.

Existem vários tipos de seleção. Um dos métodos utilizado para este fim é o método roleta, muito utilizado entre a maioria dos pesquisadores de AGs. Metaforicamente, cada indivíduo da população ocupa uma porção da roleta, proporcional ao seu índice de aptidão, ou seja, os indivíduos que possuírem maior aptidão ocuparão uma maior porção do que aqueles com menor aptidão. A roleta, então, assim como mostra a Figura 2, é girada e o indivíduo escolhido é aquele em que a seta parar sobre ele (FILITTO, 2008).

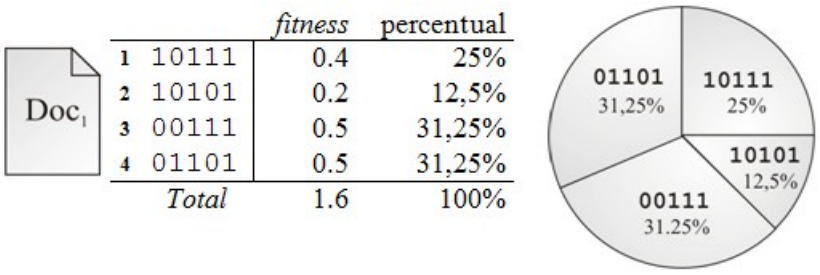

Figura 2 – Demonstração do método de seleção Roleta **Fonte:** "http://www.dgz.org.br/fev09/Art04.htm".

2.1.2.4 Cruzamento

Segundo Linden (2012), após a seleção dos pais, irá ocorrer então o processo de cruzamento ou *crossover* e, como no cruzamento natural, neste processo dois novos indivíduos, ou seja, duas novas soluções são formadas a partir de características daquelas que se cruzaram, ou seja, serão geradas duas novas soluções que conterão alguns cromossomos de uma solução e alguns cromossomos de outra.

Para Lacerda e Carvalho (2015), o operador *crossover* é um mecanismo de busca dos AGs para explorar regiões desconhecidas do espaço de busca. Ele é aplicado a cada par de cromossomos dos indivíduos que passaram pelo processo de seleção, gerando assim, dois novos indivíduos.

Para Filitto (2008), o cruzamento combina os cromossomos pais a fim de gerar cromossomos filhos e para isso existem vários tipos de cruzamento.

Um dos cruzamentos muito utilizado é o cruzamento de um ponto, que divide a lista de cromossomos selecionados em um ponto de sua cadeia, esse ponto é escolhido aleatoriamente. Após essa divisão, é copiada uma parte dos cromossomos de cada pai para definir os cromossomos dos indivíduos filhos. Neste método, é comum os pais gerarem dois novos filhos (FILITTO, 2008).

Um outro cruzamento muito utilizado é o cruzamento uniforme, que ainda segundo o autor, gera cada gene do descendente copiando o gene em questão de um dos pais, em que se usa uma máscara de cruzamento que é gerada aleatoriamente para fazer a escolha. Para criar cada cromossomo do novo indivíduo é feita uma iteração em todas as posições da máscara fazendo uma análise dos seus valores, quando o valor da posição for 1, o gene do primeiro pai referente à mesma posição da máscara é copiado, se o valor for 0 será copiado o gene do segundo pai, depois desse processo é gerado o novo descendente.

Segundo Lacerda e Carvalho (2015), o tipo de cruzamento uniforme difere do cruzamento de um ponto uma vez que este sempre leva à metade dos *bits* de cada pai.

2.1.2.5 Mutação

Para Linden (2012), também pode ocorrer a mutação em que, como ocorre na natureza, aleatoriamente o valor dos cromossomos de um indivíduo pode ser alterado. A mutação ocorre de acordo com uma taxa definida. Basicamente é definida uma porcentagem baixa e então um número de 0 a 1 é sorteado e multiplicado por 100, se o resultado for menor que a porcentagem definida, irá ocorrer a mutação para aquele indivíduo.

Para Lacerda e Carvalho (2015), a mutação melhora a diversidade dos cromossomos na população. Em contrapartida, depois de realizada a mutação se perdem as informações contidas no cromossomo. Assim, para assegurar a diversidade deve-se usar uma taxa de mutação pequena.

Assim como no cruzamento há vários tipos de mutação, dentre eles a mutação de *bit* que é um tipo de mutação mais simples de ser implementada.

Segundo Filitto (2008) este é o operador mais fácil de trabalhar podendo ser aplicado em qualquer forma de representação binária dos cromossomos. Este tipo de mutação gera uma probabilidade de mutação para cada *bit* do cromossomo, caso a probabilidade sorteada estiver dentro da taxa de mutação definida, o *bit* sofrerá mutação, recebendo um valor determinado de forma aleatória dentre os valores que podem ser assumidos pelo cromossomo.

No problema a ser solucionado na aplicação deste trabalho, há um cenário em que existem diversas soluções para se resolver o problema e deseja-se encontrar a melhor dentre elas. Conforme descrito anteriormente, os AGs são uma das melhores opções para se resolver este tipo de problema, por este motivo esta técnica foi escolhida.

2.2 Tecnologias

Abaixo serão listadas as tecnologias que serão utilizadas no desenvolvimento do projeto.

2.2.1 Linguagem de programação Java

Segundo Oracle (2015b), a Linguagem Java foi projetada para permitir o desenvolvimento de aplicações seguras, portáteis e de alto desempenho para a mais ampla gama de plataformas de computação.

De acordo com Schildt (2007), o Java foi criado em 1991 pela *Sun Microsystems* e foi baseado em uma linguagem já existente, o C++, que foi escolhida por ser orientada a objetos e por gerar códigos compactados, o que era exatamente o que eles precisavam para implantar em pequenos aparelhos. Além dessas características, um requisito desejável era que a nova linguagem fosse independente de plataforma, para que fosse executado em qualquer arquitetura, tais como, TVs, telefones, entre outros, e então, para atender esta exigência, foi criado o conceito de máquina virtual, que ficou conhecido como *Java Virtual Machine* - JVM[2].

O ponto chave que permite o Java resolver o problema de portabilidade é o fato de o código ser compilado em *Bytecode*, que é um conjunto genérico de instruções altamente otimizado que é executado pela JVM, e esta, por sua vez, traduz o mesmo para a arquitetura a qual

[2] O termo Java Virtual Machine será referenciado pela sigla JVM a partir deste ponto do trabalho.

ela está instalada o que possibilita a execução do programa em várias plataformas (SCHILDT, 2007).

Além da portabilidade, o Java também é *multithread*, ou seja, permite a execução de múltiplas tarefas simultaneamente. A linguagem também conta com um *automatic garbage collector* que consiste em um mecanismo de gerenciamento e limpeza de memória que a JVM acomoda. Além disso o Java suporta uma extensa biblioteca de rotinas que facilitam a interação com protocolos TCP/IP, como HTTP e FTP (SCHILDT, 2007).

De acordo com Junior (2007),

> Atualmente a linguagem está organizada em três segmentos principais:
>
> - JavaMe (*Java Micro Edition*) - Destinado a pequenos dispositivos computacionais móveis, tais como celulares, PDAs e *set-top boxes*. É composto de máquina virtual otimizada para ambientes mais restritos, com especificações de funcionalidades e uma API mais compacta;
>
> - JavaSE (*Java Standard Edition*) - Integra os elementos padrão da plataforma e permite o desenvolvimento de aplicações de pequeno e médio porte. Inclui todas as APIs consideradas de base, além da máquina virtual padrão;
>
> - JavaEE (*Java Enterprise Edition*) - Voltada para o desenvolvimento de aplicações corporativas complexas. Adiciona APIs específicas aos elementos padrão da plataforma.

Para Santos (2003), Java é uma linguagem Orientada a Objetos (OO), este paradigma usa objetos, criados a partir de modelos, também chamados de classes, para representar e processar dados em aplicações computacionais. Basicamente uma classe ou modelo é um conjunto de especificações de um objeto, ou seja, que tipo de dados este deve ter e quais operações este terá e então após a criação da classe o programador pode criar um objeto desta. Uma analogia simples seria a planta de uma casa e a construção da mesma em si, a planta representa a classe, ou seja, como a casa deve ser feita e a casa depois de construída representa o objeto.

Os dados e operações são guardados em objetos e estes são os elementos centrais do programa. Assim a aplicação como um todo é vista como uma coletânea de objetos que se relacionam uns com os outros. Cada objeto representa um conceito real de uma parte do problema. Isto permite que o desenvolvimento se torne menos complexo pois os conceitos são familiares às pessoas envolvidas no projeto pelo fato de que a aplicação não está organizada em processos estruturados mas sim em objetos que espelham o mundo real e interagem entre si (MANZONI, 2005).

Para Santos (2003), os dados pertencentes aos modelos são representados por tipos nativos, característicos das linguagem de programação e podem também ser representados por outros modelos criados pelo programador.

Para o desenvolvimento do sistema de informação deste projeto, o paradigma de orientação a objetos, que será implementado na linguagem Java, será imprescindível devido à alta complexidade do problema a ser resolvido. Pois cada objeto representará partes do algoritmo genético que será desenvolvido, tais como a representação do Indivíduo, Cromossomos, etc, o que facilitará muito o desenvolvimento.

O produto resultante deste trabalho faz uso do Java EE e, como já foi dito em seções anteriores, será implementado em plataforma WEB. Para tal, faz-se necessário o uso de um servidor de aplicações WEB. Dentre as opções disponíveis, foi escolhido o *Tomcat* pelo fato de este ser o mais simples e ter os atributos mínimos necessários para a aplicação que será desenvolvida.

Segundo Vukotic e Goodwill (2011), o *Tomcat* é um servidor *open source* e *container* de aplicações *web* baseados em Java. Ele foi criado para executar *servlets*, que também é uma denominação para classe dentro da especificação do Java para WEB, e arquivos de marcação que definem os elementos visuais da tela que será explanado na seção. O *Tomcat* foi criado como um subprojeto da *Apache-Jakarta*, porém como ficou muito popular entre os desenvolvedores, a *Apache* o denominou como um projeto separado e vem sendo melhorado e apoiado por um grupo de voluntários da comunidade *Java Open Source*, que o faz uma excelente solução para desenvolvimento de uma aplicação *web* completa.

2.2.2 Interface Gráfica - JSF

Para a construção da interface gráfica (páginas web) da aplicação desenvolvida neste trabalho, foi utilizado um *framework* nativo nas novas versões do Java denominado *Java Server Faces* (JSF).

Bergsten (2004) afirma que o JSF é um *framework server-side* baseado em componentes *web*, cuja principal função é abstrair os detalhes de manipulação dos eventos e organização dos componentes na página *web*. Por meio dele, é possível desenvolver páginas mais sofisticadas de forma simples, abstraindo, inclusive, o tratamento de requisições e respostas. Isto permite ao desenvolvedor focar-se no *back-end* da aplicação, ou seja, na lógica, e não se preocupar com detalhes a respeito de requisições e respostas HTTP e como obter as informações recebidas e/ou enviadas através deste protocolo.

De acordo com Oracle (2015a), o JSF é de fácil aprendizado e utilização, pois possui sua arquitetura claramente definida, sendo dividida entre a lógica da aplicação e apresentação. Esta

divisão é possível pois ele utiliza o padrão de projeto *Model-View-Controller* - MVC[3], tornando-o um importante *framework* para desenvolvimento de aplicações utilizando a plataforma Java *Web*.

Segundo Gamma et al. (2009), o padrão de projeto MVC é dividido em três partes. O *Model* é a lógica e acesso aos dados da aplicação, a *View* é camada de apresentação e por último o *Controller* é responsável por definir a interface entre a lógica e a apresentação. Portanto, todo tipo de requisição ou resposta deve ser obrigatoriamente enviada ao *Controller*, que, por sua vez encaminhará para a camada de visão ou de lógica. A Figura 3 demonstra um exemplo do modelo MVC utilizando o JSF.

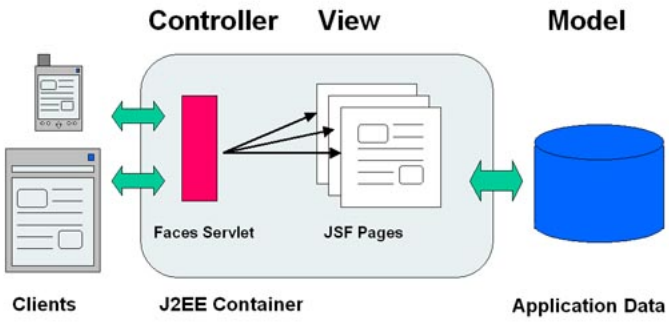

Figura 3 – Demonstração do Modelo MVC. **Fonte:** http://www.javabeat.net/jsf-2/.

Ao utilizar o JSF, toda e qualquer interação que o usuário realizar com a aplicação será executada por um *servlet* chamado *Faces Servlet*. Ele é a responsável por receber tais requisições da camada de visão e redirecioná-las à lógica da aplicação e, posteriormente, enviar a resposta ao usuário (FARIA, 2013).

Segundo Lucknow e Melo (2010), a especificação do *JavaServer Faces* foi definida pela *Java Comunity Process* - JCP[4], que é uma entidade cujo objetivo é especificar a evolução do Java. E por este motivo grandes empresas como *Apache*, *IBM*, *Oracle*, entre outras, participaram desta definição. As implementações mais conhecidas são da especificação do JSF são:

- *Sun Mojarra* (antes JSF R1) – implementação de referência (<http://javaserverfaces.java.net/>)
- *MyFaces* da *Apache* (<http://myfaces.apache.org/>)

Com essas implementações é possível utilizar todos os recursos do padrão JSF, como formulários, tabelas, *layout*, conversão e validação de eventos, além de toda a inteligência

[3] MVC: *Model-View-Controller - Design pattern*, padrão de arquitetura de software que separa a informação (e as suas regras de negócio) da interface com a qual o usuário interage.

[4] O termo *Java Comunity Process* será referenciado pela sigla JCP a partir deste ponto do trabalho.

para a interpretação dos arquivos de configuração e interação com o *container* Java. Como o JSF é um padrão de mercado, várias empresas investem no desenvolvimento de bibliotecas de componentes como, calendário, barras de progresso, menus, efeitos de transição entre outros (LUCKNOW; MELO, 2010).

Algumas das principais bibliotecas de componentes são:

- *Trinidad*, da *Apache MyFaces* (<http://myfaces.apache.org/trinidad/>);
- *Tobago*, da *Apache MyFaces* (<http://myfaces.apache.org/tobago/>);
- *ICEFaces*, da *ICESoft* (<http://www.icefaces.org/>);
- *RichFaces*, da *JBoss* (<http://www.jboss.org/richfaces/>);
- *Tomahawk*, da *Apache MyFaces* (<http://myfaces.apache.org/tomahawk/>);
- *PrimeFaces* (<http://www.primefaces.org/>).

Neste trabalho, é utilizada a biblioteca PrimeFaces, que é uma biblioteca de componentes que implementa a especificação do JSF. Ele possui uma ampla gama de componentes disponíveis para auxiliar o desenvolvimento de interfaces *web* ricas, além de possuir o seu código fonte aberto (ROSS; BORSOI, 2012).

Para Juneau (2014), uma das grandes vantagens do Primefaces é a facilidade de integração entre ele e o JSF, bastando apenas incluir a biblioteca do Primefaces no projeto JSF, salvo alguns componentes específicos, como o *file upload*, que necessita de pequenas configurações adicionais. Essas mudanças, quando necessárias, devem ser realizadas no arquivo de configuração da aplicação que, por padrão, é chamado de *web.xml*, porém, o mesmo pode ser alterado pelo desenvolvedor.

Por possuir as vantagens descritas acima e uma simples configuração, além de ser um *framework cross-browser*[5], o JSF foi escolhido para auxiliar no desenvolvimento das páginas *web* deste trabalho.

2.2.3 Armazenamento de dados

A aplicação desenvolvida neste trabalho irá demandar o armazenamento de informações das costureiras e da disponibilidade das mesmas. Além disso, as saídas oferecidas pelo algo-

[5] *Cross-Browser* - Compatibilidade com todos os tipos de dispositivos e navegadores

ritmo genético deverão ser armazenadas para uso posterior, alimentando, assim, a base de dados do sistema, fazendo-se necessário o uso de um banco de dados.

O gerenciador de banco de dados a ser utilizado nesta aplicação é o PostgreSQL, que foi escolhido por ser uma ferramenta robusta e *open source*. O PostgreSQL é um banco de dados relacional desenvolvido pela universidade da California por volta de 1970. Na época, o projeto se chamava Ingres e só passou a se chamar Postgres por volta de 1986 quando Michael Stonebraker adicionou o conceito de orientação a objetos ao projeto e decidiu então definir um novo nome para a nova versão (DOUGLAS; DOUGLAS, 2003).

Para Date (2004) , "um banco de dados é um sistema computadorizado cuja funcionalidade geral é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando as solicitar". Já o conceito de banco de dados relacional é definido por Price (2008, p.30) como:

> uma coleção de informações relacionadas, organizadas em tabelas. Cada tabela armazena dados em linhas; os dados são organizados em colunas. As tabelas são armazenadas em esquemas de banco de dados, que são áreas onde os usuários podem armazenar suas próprias tabelas.

Para manipular e acessar as informações em um banco de dados relacional é usada uma linguagem denominada SQL (*Structured Query Language*), que foi projetada especificamente para este fim. A linguagem foi desenvolvida pela IBM por volta de 1970 que tomou como base o trabalho do Dr. Edgar Frank Codd e possui uma sintaxe simples de fácil aprendizado e utilização (PRICE, 2008).

3 QUADRO METODOLÓGICO

O quadro metodológico descreve os passos realizados para a execução do trabalho. Neste capítulo serão listados, em suas seções, os itens essenciais no desenvolvimento do trabalho, sendo eles as técnicas, procedimentos, práticas e instrumentos utilizados, o contexto de aplicação e o tipo de pesquisa.

3.1 Tipo de pesquisa

Para Gil (1999, p.42), a pesquisa tem um caráter pragmático, é um “processo formal e sistemático de desenvolvimento do método científico. O objetivo fundamental da pesquisa é descobrir respostas para problemas mediante o emprego de procedimentos científicos”.

Este trabalho terá como base a metodologia de pesquisa aplicada, pois será desenvolvida uma aplicação inteligente utilizando Algoritmos Genéticos para o auxílio na tomada de decisão sobre o processo de produção de calças de uma confecção. Esta pesquisa consiste em procurar respostas para problemas propostos baseados em padrões e conhecimentos já existentes.

Gerhardt e Silveira (2009, p.35) afirmam que o método de pesquisa aplicada, "objetiva gerar conhecimentos para aplicação prática, dirigidos a solução de problemas específicos. Envolve verdades e interesses locais."

Segundo Zanella (2009), a pesquisa aplicada tem como motivação básica a solução de problemas concretos, práticos e operacionais e também pode ser chamada de pesquisa empírica, pois o pesquisador precisa ir a campo, conversar com pessoas e presenciar relações sociais.

Como citam Marconi e Lakatos (2009), a pesquisa aplicada caracteriza-se por possuir um interesse prático, quando os resultados serão aplicados ou utilizados na solução de problemas que ocorrem na realidade, sempre visando gerar conhecimento para solucionar situações específicas.

Como já foi explicado o tipo de pesquisa em que se enquadra este trabalho, ela deve ser aplicada a um determinado contexto, conforme será explicado na seção a seguir.

3.2 Contexto de pesquisa

Sabe-se que com a alta competitividade no mercado, as empresas, cada vez mais, buscam diferenciais para seus produtos e, neste cenário, a ideia de redução de custos se torna essencial, uma vez que tal redução pode ser refletida no preço dos produtos permitindo que estes se diferenciem dos demais. Dentre os fatores que viabilizam tais reduções está a otimização de processos que consiste em organizar os procedimentos relacionados à produção de forma que estes se tornem mais eficazes.

O software desenvolvido neste trabalho visa organizar uma linha de produção de forma que esta se torne o mais eficiente possível. Será utilizada como base uma fábrica de confecção de calças situada na cidade de Cachoeira de Minas - MG, porém a base de conhecimento pode ser aplicada a outros tipos de negócios que seguem o mesmo padrão de desenvolvimento de produtos.

Como cada funcionário trabalha em sua casa, é preciso ter uma boa forma de distribuir o trabalho, permitindo que a produção possa ser feita em um tempo menor e com custo reduzido. Para isso, é necessário que o software faça a distribuição dos lotes a serem confeccionados entre as costureiras de forma que estas sejam alocadas de forma a receberem a quantidade de peças de acordo com sua capacidade de trabalho, considerando também o tempo que cada costureira irá gastar para pegar as peças e o material necessário para realizar seu trabalho, além disso o software considera o preço cobrado por cada uma das delas.

A aplicação cruza então todas estas informações de forma a se produzir soluções para o problema e, no final, a melhor destas soluções será aquela que ofereça o menor custo e o menor tempo de produção. Porém, sabe-se que, em algoritmos genéticos, não há garantias que a solução encontrada é a melhor para se resolver o problema mas a tendência é que a solução proposta pelo algoritmo seja muito boa e seria muito difícil encontrá-la manualmente.

3.3 Instrumentos

Segundo Faria (2015), instrumentos de pesquisa são a forma pela qual os dados são coletados para a realização do trabalho, podendo ser, dentre outras, por meio de reuniões, questionários e entrevistas. Para este trabalho foi utilizado os instrumentos descritos nas subseções a seguir.

3.3.1 Entrevistas

Segundo Haguette (1997), entrevista é uma interação entre duas pessoas em que uma representa o entrevistador, que através de perguntas, obtêm informações por parte de outra pessoa que representa o entrevistado.

Foi realizada uma entrevista com o dono da empresa de confecção com o objetivo de entender seu modelo de negócio para que então fosse possível começar a fazer o levantamento dos requisitos do sistema. Para Pressman (2011, p.128), levantamento de requisitos de software consiste em

> "perguntar ao cliente, aos usuários e aos demais interessados quais são os objetivos para o sistema ou produto, o que deve ser alcançado, como o sistema ou produto atenda às necessidades da empresa e, por fim, como o sistema ou produto deve ser utilizado no dia a dia".

A entrevista ocorreu no dia 09/05/2015 para conhecer mais sobre o processo de produção da fábrica. Nesta entrevista, ficou esclarecido todo processo e também se teve acesso à forma como era controlada a distribuição da produção entre os funcionários. Toda produção era controlada por meio de planilhas Excel, gerenciadas e alimentadas pelo proprietário. Tais planilhas contemplavam as estimativas de produção, as datas de entrega dos lotes encomendados, levando em consideração a quantidade de peças por lote, o corte e também o tempo que cada lote levaria para ser entregue.

3.3.2 Reuniões

De acordo com Carvalho (2012), reunião é o ajuntamento de pessoas para se tratar de um determinado assunto em que é necessário que se tenha conclusões sobre as questões que foram discutidas.

Durante o desenvolvimento do trabalho seriam realizadas várias reuniões com o proprietário da fábrica de calças para sanar as dúvidas, sugestões e outros assuntos que poderiam surgir. Todavia foi realizada apenas uma reunião com o proprietário da empresa para poder entender como funciona o processo de produção, pois foi constatado nesta reunião que o processo de produção havia sido alterado. No processo inicial, o qual foi a base para este trabalho, cada funcionário trabalhava em sua residência e as peças eram distribuídas entre eles. Atualmente o processo passou por muitas mudanças, uma delas é que a produção é feita em um lugar apenas,

sem a necessidade de transportar as peças entre as casas dos funcionários. Segundo o proprietário isso gerou um ganho de tempo bem expressivo, pois as peças circulavam dentro de um mesmo local e não pela cidade.

Considerando essa mudança, não foram feitas outras reuniões com o proprietário da fábrica, pois o trabalho não atende mais o processo de produção atual da fábrica, porém o mesmo pode ser usado em outras empresas que seguem a forma de produção pesquisada. Assim, foi definido um escopo para o desenvolvimento da aplicação baseando-se no processo inicial da fábrica de calças, que se resume em construir uma aplicação levando em consideração que:

- o processo de desenvolvimento das calças deveria ser dividido em atividades com ordem de precedência;
- cada atividade poderia ser feita por uma ou mais costureiras, de acordo com a habilidade de cada uma;
- cada costureira gasta um tempo, medido em segundos, para fabricar uma peça;
- cada costureira cobra um preço por peça produzida;
- o usuário deveria ser capaz de cadastrar um novo processo, costureiras e habilidades;
- o usuário deveria cadastrar o tempo e o preço de produção, por peça, para cada costureira além de sua posição geográfica;
- o total de peças deveria ser dividido em lotes e cada costureira deveria receber uma quantidade de lotes distribuída aleatoriamente;
- o usuário deveria informar uma data de entrega das peças de um processo e uma data de início de execução do mesmo para que se pudesse calcular o prazo;
- o software deveria então oferecer como saída a melhor distribuição de forma a se produzir no menor tempo dentro do prazo, com o menor custo, considerando o tempo e o preço de produção de cada costureira e o tempo de transporte das peças entre elas.
- se o software não conseguisse encontrar nenhuma solução abaixo do prazo, a busca pelo menor custo seria ignorada e o menor tempo encontrado seria retornado.

A seção a seguir descreve como foi definido o escopo demonstrado acima.

3.4 Procedimentos

Esta seção descreve os procedimentos realizados na execução do trabalho, definindo primeiramente o *framework* de desenvolvimento e explicando como o algoritmo genético foi desenvolvido para os requisitos definidos no escopo.

3.4.1 Framework de desenvolvimento

Primeiramente é necessário ressaltar que, para o desenvolvimento da aplicação, foi utilizada uma base desenvolvida pelo professor Artur Barbosa durante as aulas de sistemas especialistas, do VII período do curso de sistemas de informação nesta universidade. Esta base também denominada *framework*, define regras a serem seguidas no desenvolvimento de cada elemento de um algoritmo genético. Tal *framework* é definido dentro da seguinte estrutura:

- Classe `GAModel`:

  A classe `GAModel` é basicamente a classe mãe de todos os elementos de um algoritmo genético, que representa o modelo que irá armazenar a população de indivíduos além de ser a classe que armazena os parâmetros que definem as configurações do algoritmo, tais como, tipo de cruzamento, tipo de mutação, tamanho da população etc.

  A classe contém os seguintes atributos:

  - *populationSize*: este atributo define qual será o tamanho da população, ou seja, quantos indivíduos irão formar cada população;
  - *generationQuantity*: como já explicado no quadro teórico, o processo de cruzamento e mutação se repete até que o número de indivíduos, definido no atributo anterior, seja atingido formando assim uma nova população e então, por sua vez, este processo de geração de novas populações se repete até que seja atingido um número de gerações definido pelo programador. Este atributo representa esta quantidade;
  - *elitism*: atributo do tipo *boolean* que representa se o algoritmo vai ter a função de elitismo. Esta função, como já foi explicada no quadro teórico, quando está ativada (com valor *true*), no momento de começar a se criar uma nova população os dois melhores indivíduos da população que será substituída já começam a fazer parte da nova população, antes de começar o processo de cruzamento e mutação. Este

mecanismo garante que a nova população terá pelo menos dois indivíduos iguais ao da antiga população, o que irá impedir que a nova população não seja pior que a anterior;

- *foreignIndividualRate*: este atributo define a taxa de novos indivíduos que devem entrar em novas populações e será visto com mais detalhes na seção Indivíduos Estrangeiros.
- *mutationRate*: como descrito no quadro teórico, a mutação é o fato de realizar pequenas alterações no indivíduo a fim de que este possa se tornar ainda melhor. Este parâmetro define uma porcentagem, geralmente baixa, que define quando o indivíduo sofrerá mutação ou não. Esta questão ficará mais clara logo abaixo, quando será explicado o passo-a-passo da execução do algoritmo.
- *mutationQuantity*: caso a mutação for ocorrer para o indivíduo, a alteração aleatória será feita nos cromossomos. Este parâmetro define quantos cromossomos do indivíduo deve ser alterado pela mutação;
- *selectionType*: conforme descrito no quadro teórico, existem várias formas de seleção dos indivíduos para realizarem o cruzamento. Este parâmetro define qual será a forma escolhida pelo programador ao implementar o seu problema.No *framework* este parâmetro é do tipo `enum` e pode assumir 2 valores o *ROULETTE*, que define que o método de seleção utilizado será o de roleta e o *CLASSIFICATION*, que define que o método a ser utilizado é o de classificação. Neste projeto o método padrão que já foi pré-definido no código foi o método de roleta;
- *crossType*: Assim como a seleção, existe diversas formas de fazer o cruzamento dos indivíduos. Este atributo, também do tipo `enum` representa a forma de cruzamento e pode receber os valores *Binary*, *Permutation*, *Uniform* e *Aritmetic*, esses valores definem qual implementação de cruzamento o algoritmo deve utilizar, esta aplicação utilizará o método *Permutation*, desta forma somente um método de cruzamento foi implementado conforme mostra a seção Seleção, cruzamento e mutação;
- *mutationType*: Segue a mesma forma que o selectionType e o crossType e pode assumir os valores *Permutation*, *Binary* e *Numerical*, e neste projeto foi o escolhido o método *Permutation*.

- Classe `Individual`:

A classe abstrata `Individual` representa a estrutura básica de um indivíduo. A classe contém uma `lista` do tipo `Cromossomo`, que será descrita posteriormente, que contém uma coleção de objetos que representam as características da solução.

A Classe contém ainda um atributo chamado `valor` que irá armazenar a qualidade, ou seja, qual é o custo da solução representada pelo indivíduo, tal valor é recebido no retorno da operação `calculateValue()` descrita abaixo.

Com relação as operações, além dos *getters and setters*, a classe contém a operação abstrata `calculateValue()`, que realiza a função de avaliação, que mede a qualidade do indivíduo. Desta forma, ao utilizar este *framework*, a classe que representa o indivíduo do problema deve herdar desta classe `Individual`. Fazendo isso, tal classe passará a ter uma lista de cromossomos e o atributo que representa o seu valor e a classe obrigatoriamente terá que implementar a operação `calculateValue()`, permitindo assim que o programador desenvolva a função de avaliação específica para o seu problema.

- Classe `Chromosome`:

  É uma classe abstrata, que possui todos os métodos abstratos, desta forma ela só existe para garantir que os cromossomos do problema irão implementar os métodos necessários para o funcionamento do algoritmo. Estes métodos são:

  - `equals`: necessário para efeito de comparação dos cromossomos;
  - `doMutation`: realiza a mutação. Este deve ser implementado pela classe que representa o cromossomo, pois a mutação é feita nos cromossomos.
  - `clone`: devolve um objeto exatamente com os mesmos atributos do objeto, porém em uma instância diferente.

- Classe `IndividualPair`:

  A classe `IndvidualPair` possui uma estrutura simples. Apenas representa dois indivíduos. Ela se torna necessária, pois o processo de cruzamento dos indivíduos retornam dois novos indivíduos, desta forma, como no Java não é possível retornar dois valores, é retornado então um objeto desta classe contendo os dois novos indivíduos criados.

- Classe `GAController`:

  A classe `GAController`, como o próprio nome já diz, é o controlador de todo processo de execução do algoritmo genético. Ela recebe no seu construtor o modelo que é do tipo

`GAModel`, que como já explicado anteriormente, armazena os parâmetros a serem seguidos na execução do algoritmo. Além disso, através do seu método principal denominado `execute()`, ela é responsável por criar novas populações, a partir de cruzamentos, mutações, elitismo, etc, tendo também a responsabilidade de chamar a função de classificação e avaliação de cada indivíduo.

Os pontos a seguir descrevem basicamente os passos executados dentro do método execute(). Outros detalhes serão vistos mais adiante quando será descrita a implementação do algoritmo da fábrica de calças, pois será necessário realizar algumas adaptações neste *framework*.

- Criação da população inicial, através do método createInitialPopulation() do objeto que será extendido da classe GAModel;
- Classificação e avaliação da população inicial através do método `classify()`;
- Realização do processo de elitismo, através do método `doElitism()`;
- Inserção de indivíduos estrangeiros na população;
- Realização do processo de seleção de indivíduos, através do método `doSelection()`;
- Execução do processo de cruzamento e mutação, através do método `doCrossing()` e `doMutaion()` respectivamente.

Após estes passos, uma nova população foi criada e está armazenada na variável newGeneration, assim o método `setPopulation()` do objeto `model` é chamado para então substituir a antiga população pela nova. Como a execução está dentro de uma estrutura de repetição `for`, ocorre um `loop` e então é recomeçado o processo de criação de uma outra população. Este processo para quando o número de gerações, definido no parâmetro `generationQuantity` do objeto `model` for atingido, neste caso é dado o comando `break` e o `loop` é encerrado.

As próximas seções apresentam a implementação dos requisitos da aplicação e dos elementos do algoritmo genético, seguindo as definições do *framework*.

3.4.2 Representação do processo de produção

Primeiramente foi definido como seria o processo de fabricação. Este foi pensado com base no processo da fábrica, em que a confecção das peças deveria ser dividida em atividade-

des que representam cada parte da calça. Neste contexto, surgiu a necessidade de determinar uma ordem para a execução do processo, devido ao fato de que algumas atividades dependem da finalização de outras para poderem ser realizadas. A Figura 4 demonstra basicamente um exemplo de ordem de execução do processo de confecção.

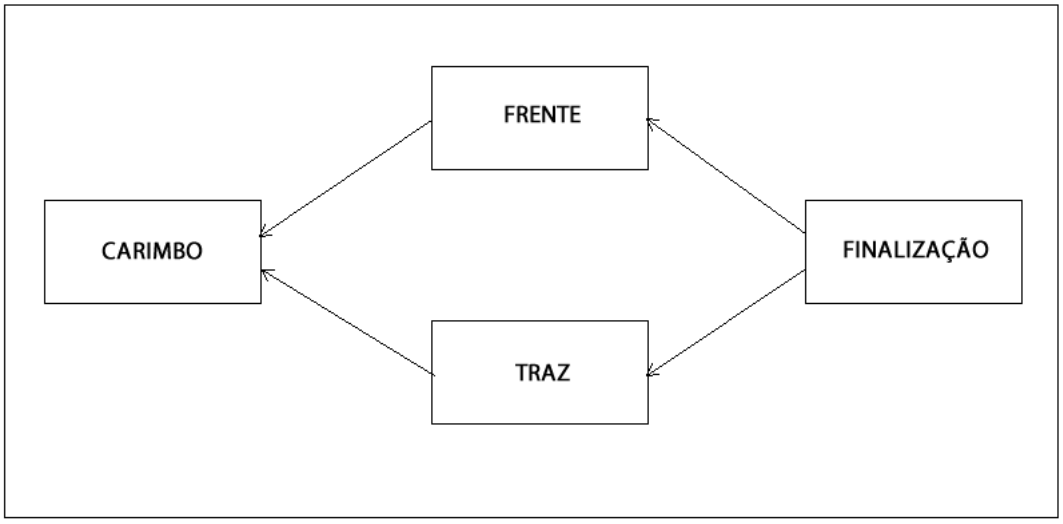

Figura 4 – Demonstração de um processo de fabricação **Fonte:** Desenvolvido pelos autores.

Uma questão importante que foi definida é que, independente da complexidade e tamanho do processo, este deve sempre começar com a atividade Carimbo e finalizar com a atividade Finalização. Isso ocorre pois a Finalização é sempre a última atividade de qualquer processo da fábrica e o Carimbo é uma atividade simbólica que representa o fato de o dono da fábrica separar o material de costura. O tempo e o custo gastos nesta separação não são contabilizados no algoritmo genético, somente o tempo de transporte dos materiais até as costureiras são considerados na distribuição, além disto esta atividade terá somente uma pessoa trabalhando, que neste caso será o Marcelo, dono da fábrica.

Para armazenar este processo e suas atividades no software, foram utilizadas tabelas do banco de dados, conforme mostra a Figura 5.

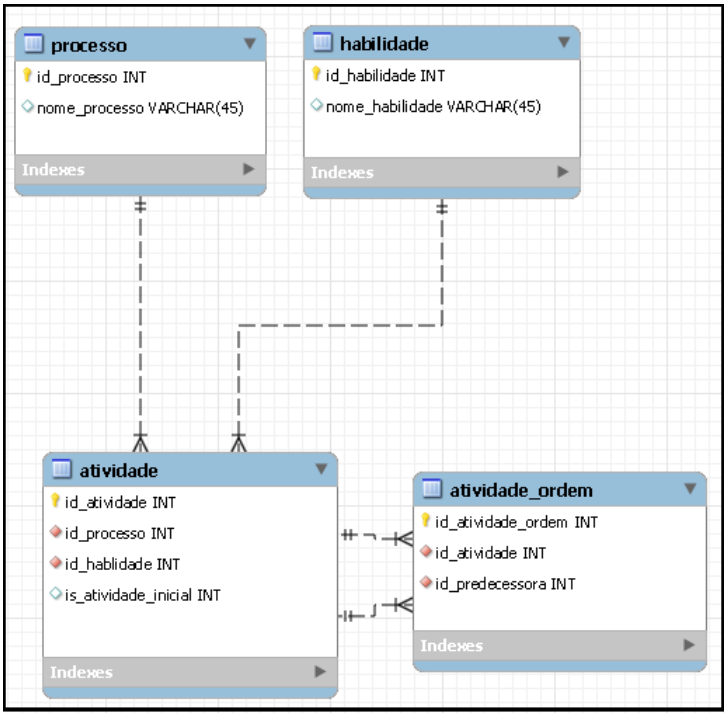

Figura 5 – Representação do processo de fabricação no banco de dados **Fonte:** Desenvolvido pelos autores.

A tabela `processo` tem como finalidade gerar um código único para representar cada processo de produção, cada processo representa um modelo, cada modelo de calça possui um processo diferente que pode possuir diferentes atividades que são representadas na tabela `atividade`, onde é feita a relação que define quais são as atividades de um processo, além de conter quais são as habilidades necessárias para cada atividade, ou seja, cada registro desta tabela representa uma atividade do processo e qual habilidade é necessária para sua execução. O campo `is_atividade_inicial`, quando tem o valor 1, define que tal atividade é a atividade Finalização. Este *flag* é importante no momento de calcular o tempo total de execução do processo e será visto com mais detalhes posteriormente e, por fim, a tabela `atividade_ordem` é onde é feita a definição de ordem de execução das atividades.

A Figura 6 demonstra o mapeamento da relação entre a tabela `atividade` e atividade_ordem para o Java.

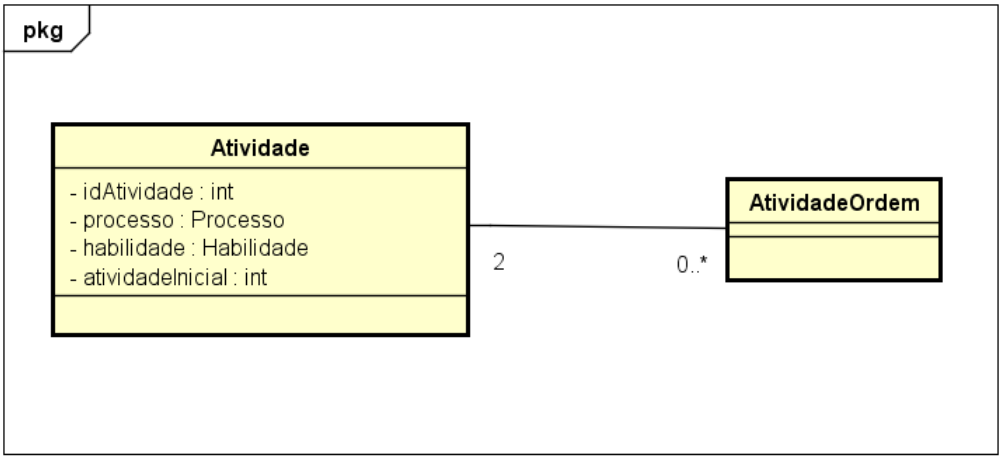

Figura 6 – Classes Atividade e AtividadeOrdem **Fonte:** Desenvolvido pelos autores.

A classe Atividade pode ter zero ou vários objetos da classe AtividadeOrdem. Esta possui dois objetos da própria classe Atividade, um representando uma atividade e outro representando a sua predecessora.

Considerando o processo de produção e o modelo de dados apresentados acima, foi desenvolvido o algoritmo genético para a solução do problema. Conforme já demonstrado no Quadro Teórico, um algoritmo genético deve seguir uma ordem de execução conforme mostra a Figura 7.

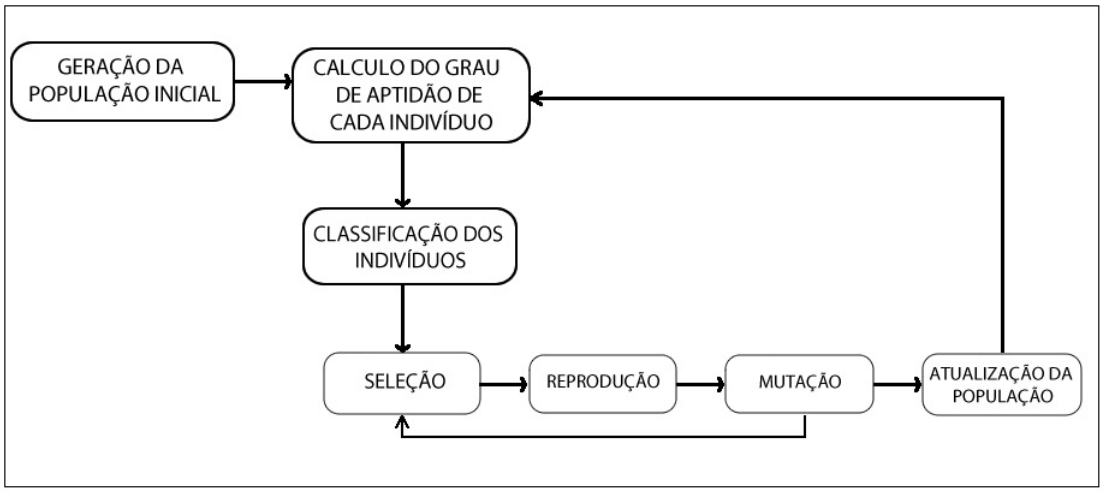

Figura 7 – Demonstração da execução de um AG **Fonte:** Desenvolvido pelos autores.

O primeiro passo para a execução do algoritmo então foi o desenvolvimento de uma lógica para a criação da população inicial de indivíduos, conforme explicado na próxima seção.

3.4.3 População inicial: Distribuição das atividades, Indivíduos e Cromossomos

O processo de criação da população inicial assim como o processo de seleção, cruzamento e mutação e a chamada da função de avaliação de cada indivíduo, ocorre dentro da classe de controle do *Framework*, ou seja, tal classe faz a orquestração de toda a execução do algoritmo genético e é denominada `GAController`.

A classe recebe em seu construtor um objeto de `GAModel` com as configurações do algoritmo, porém, como esta classe é abstrata, foi criado a classe `ProcessoModel` que herda de `GAModel`, passando a ter todos os atributos da classe mãe. Para gerenciar então a execução do algoritmo foi criado uma classe de serviço denominada `GeneticAlgorithmManagement`, tal classe é instanciada pelo controlador da tela de distribuição de atividades, que será explanado posteriormente, recebe, em seu método `iniciarDistribuicao`, os dados informados pelo usuário e inicia o algoritmo genético conforme mostra o Código 2.1.

Código 3.1 – Método `iniciarDistribuicao`. **Fonte:** Elaborado pelos autores.

```
1
2  public ProcessoIndividual iniciarDistribuicao(
3        int numeroLote,BigDecimal prazEmSegundos,
4        int pecasPorLote,int idProcesso){
5
6      EntityManager manager = ConFactory.getConn();
7
8      ProcessoModel model = new ProcessoModel(manager,idProcesso);
9      model.setNumeroLote(numeroLote);
10     model.setPecasPorLote(pecasPorLote);
11     model.setPrazoEmSegundos(prazEmSegundos);
12     model.setGenerationQuantity(10000);
13     model.setPopulationSize(80);
14     model.setElitism(true);
15     model.setSelectionType(GAModel.SelectionType.CLASSIFICATION);
16     model.setCrossType(CrossType.PERMUTATION);
17     model.setForeignIndividualRate(0.3f);
18     model.setMutation(GAModel.MutationType.PERMUTATION);
19     model.setMutationRate(0.05f);
20     model.setMutationQuantity(1);
21
22     GAController controller = new GAController(model);
23     return (ProcessoIndividual) controller.execute();
24 }
```

Tal método retorna um objeto do tipo `ProcessoIndividual` para o controlador da tela de distribuição, este é o melhor indivíduo encontrado, ou seja, a melhor solução encontrada,

isto será explicado com mais detalhes nas próximas seções assim como o significado de cada parâmetro do método `iniciarDistribuicao`.

A classe `ProcessoModel` é a primeira a ser instanciada e recebe em seu construtor uma conexão para o banco de dados e o `ID` do processo a qual será executado o algoritmo. O construtor desta então chama o método `getInformacoesCostureiras()` que tem por finalidade buscar no banco de dados todas as atividades do processo em questão, buscar todas as costureiras que tem a habilidade de fazer cada uma destas e criar um `HashMap` que possui como chave o `ID` da atividade e como valor uma lista de `CostureiraHabilidade`, feito isto o método também recupera a atividade final (Finalização), conforme demonstra a Código 2.2.

Código 3.2 – Método `getInformacoesCostureiras`. **Fonte:** Elaborado pelos autores.

```
1
2  public void getInformacoesCostureiras() {
3    List<CostureiraHabilidade> costureirasHabilidades = null;
4
5    if (atividadesCostureiras != null && atividadesProcesso != null){
6       atividadesCostureiras.clear();
7       atividadesProcesso.clear();
8    }
9
10   atividadesCostureiras =
11      new HashMap<Integer, List<CostureiraHabilidade>>();
12
13   atividadesProcesso =
14      atividadeDao.listAtividadesByProcesso(processo);
15
16   /* Montar um MAP tendo como chave cada atividade do processo
17      e a lista de costureiras que tenham a habilidade relacionada.*/
18   for (Atividade atividade : atividadesProcesso) {
19
20     costureirasHabilidades =
21        cdao.getCostureirasByHabilidade
22           (atividade.getHabilidade().getIdHabilidade());
23
24     atividadesCostureiras.put
25        (atividade.getIdAtividade(),costureirasHabilidades);
26
27     if (atividade.isAtividadeFinal()) atividadeFinal = atividade;
28   }
29 }
```

De acordo com o código 2.1, no objeto de `ProcessoModel` são definidas então as configurações do algoritmo genético, conforme explicado na seção *Framework* de desenvolvimento, além disso, também são definidos no objeto os parâmetros recebidos do usuário. O parâmetro `numeroLote` define qual é o número de lotes a serem produzidos no processo, o `pecasPorLote` define quantas peças cada lote possui e o parâmetro `prazEmSegundos` indica qual é o prazo em segundos, considerando a data de início informada pelo usuário e a data de entrega do processo.

Então, após a definição dos parâmetros, é criado um novo objeto de `GAController`

passando-se o objeto da classe `ProcessoModel` e logo o método `execute()` é chamado, dando-se início a execução do algoritmo genético. A primeira coisa a ser feita então é criar a população inicial de indivíduos que é criada a partir do método `createInitialPopulation()` declarado de forma abstrata na classe `GAModel` e implementado pela classe `ProcessoModel`. Tal método basicamente executa um `for` de 0 até o tamanho da população (atributo `populationSize`) e assim para cada iteração é criado um objeto de `ProcessoIndividual` passando a `atividadeFinal`, o `map` que contém as atividades e suas costureiras (`atividadesCostureiras`) criados pelo método `getInformacoesCostureiras()`, além dos atributos `numeroLote`, `pecasPorLote` e `prazoEmSegundos`, conforme mostra o código 2.3.

Código 3.3 – Método `createInitialPopulation`. **Fonte:** Elaborado pelos autores.

```
1
2 @Override
3   public void createInitialPopulation() {
4     for (int i = 0; i < getPopulationSize(); i++) {
5       population.add
6          (new ProcessoIndividual
7              (atividadeFinal, prazoEmSegundos, atividadesCostureiras,
8               this.numeroLote, this.pecasPorLote));
9   }
10 }
```

Quando se cria um novo indivíduo, a partir da instanciação de um novo objeto de `ProcessoIndividual`, acontece então a distribuição das atividades de forma a representar uma solução para o problema. A realização desta distribuição foi realizada considerando que o total de peças a ser produzido deveria ser dividido em lotes e então, em cada atividade, este número de lote deveria ser distribuído entre as costureiras que possuíssem a habilidade em questão. Por exemplo: se a quantidade total de peças de uma ordem de produção for 500, primeiramente deve-se definir qual será o número de peças por lote, neste caso, se for definido que cada lote deverá ter 50 peças, então o resultado final será 500/50 ou seja 10 lotes contendo 50 calças cada um.

Neste sentido, seguindo o exemplo apresentado na Figura 4, a distribuição deverá ser feita de forma que para cada atividade do processo seja distribuído o trabalho de 10 lotes, ou seja, 10 lotes da parte da frente deve ser confeccionado e enviados para as costureiras que sabem colocar o zíper e posteriormente estas enviam os 10 lotes para as costureiras que fazem finalização. Estas últimas irão depender também de 10 lotes da parte de trás que também devem passar pelas costureiras que fazem a confecção dos bolsos.

Com base nesses requisitos, um dos papeis desempenhados pelo algoritmo genético está na distribuição de trabalho descrita acima. Inicialmente algoritmo irá distribuir, de forma

aleatória, o número de lotes definido entre as costureiras de cada atividade, conforme mostra a Figura 7.

Finalização	Frente	Carimbo
Josi: 10	Roberta: 5	Marcelo: 10
-	Tereza: 5	-
-	Maria: 0	-

Figura 8 – Exemplo de distribuição aleatória de lotes para as costureiras **Fonte:** Desenvolvido pelos autores.

Uma costureira pode não receber lote (:0), isso permite que a decisão de quem vai participar ou não também fique por conta do algoritmo. O exemplo demonstrado está considerando que se irá produzir 500 peças em lotes de 50, resultando assim em um total de 10 lotes.

Como já explanado no quadro teórico, a estrutura do algoritmo genético é composta por populações que são formadas por indivíduos que por sua vez são formados por cromossomos. Cada indivíduo representa uma solução e cada cromossomo do indivíduo representa uma de suas características. Assim, então, é gerada uma população inicial de indivíduos e, a partir desta, um processo de cruzamento e mutação é iniciado a fim de que possam ser gerados novos indivíduos que representem soluções ainda melhores que seus antecessores.

Neste sentido, para o desenvolvimento do algoritmo de distribuição de lotes, o processo de definição de indivíduo e cromossomo foi o primeiro passo do desenvolvimento da aplicação. Isso se deve ao fato de que estes elementos compõe a parte crucial para que se possa definir a lógica a ser seguida para a definição da população inicial, o tipo de cruzamento a função de avaliação, etc.

Neste caso, cada indivíduo da população irá representar uma forma de distribuir as atividades e cada número de lotes distribuídos a determinada costureira em uma determinada atividade irá representar um cromossomo. Tomando como base o exemplo da Figura 7, o quadro, como um todo, representa o indivíduo e cada distribuição, como por exemplo a Roberta, que recebeu 5 lotes para confeccionar, representa um cromossomo.

Para fazer esta representação em Java, primeiramente foi criado uma classe denominada `ProcessoChromosome` que é representada na Figura 8:

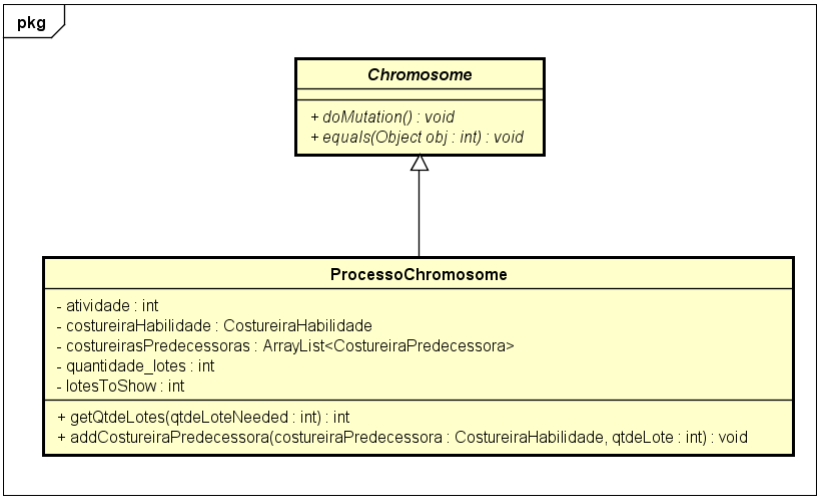

Figura 9 – Classe ProcessoChromosome **Fonte:** Desenvolvido pelos autores.

A classe `ProcessoChromosome` herda de `Chromosome` do *framework* descrito na Figura 8. Por enquanto, é necessário compreender apenas os atributos `atividade`, costureiraHabilidade e `quantidade_lotes`, que recebem seus valores pelo construtor, os demais atributos e métodos são utilizados pela função de avaliação e serão explicados mais adiante. O atributo atividade é do tipo `int` e representa o `ID` da atividade, no banco de dados, a qual se está atribuindo a costureira e a quantidade de lotes, este atributo será passado na criação de cada cromossomo sempre que for necessário se criar um novo indivíduo. O atributo `costureiraHabilidade` é do tipo CostureiraHabilidade, que representa o mapeamento da tabela costureira_habilidade do banco de dados, e faz a relação entre quais habilidades cada costureira possui e quanto tempo cada uma gasta para fazer uma peça de uma determinada parte da calça, conforme demonstra a Figura 9.

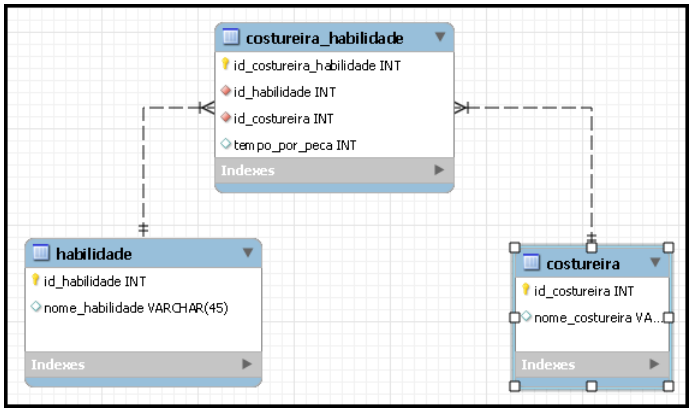

Figura 10 – Armazenamento de dados das costureiras **Fonte:** Desenvolvido pelos autores.

A classe `CostureiraHabilidade`, conforme ilustra a Figura 10, por sua vez, possui o atributo habilidade, outro que representa a costureira e um terceiro para representar o tempo

que tal costureira gasta para confeccionar uma peça com certa habilidade. Fez-se necessário ter um atributo da classe CostureiraHabilidade ao invés de simplesmente ter um objeto do tipo Costureira, pois, na função de avaliação, como será visto mais adiante, necessita ter o tempo que a costureira gasta para fazer a peça e este tempo pode variar para uma mesma costureira dependendo de suas habilidades.

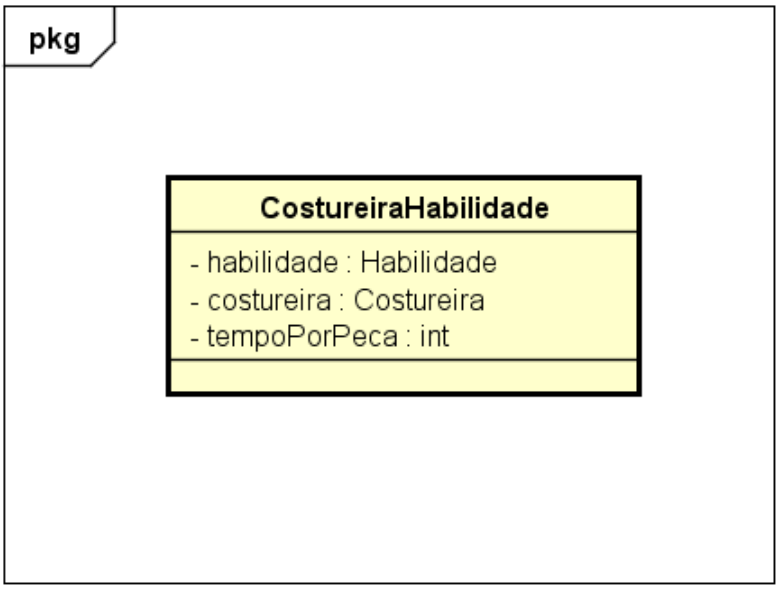

Figura 11 – Classe CostureiraHabilidade **Fonte:** Desenvolvido pelos autores.

Assim, para representar cada característica da solução, tomando como exemplo a Figura 7, o fato de Roberta fazer 5 lotes da parte da frente é representado na implementação do código criando se um objeto da classe `ProcessoChromosome` passando no construtor o `id` da atividade "Frente", um objeto de `costureiraHabilidade`, cujo atributo `costureira` represente a Marta, o atributo `habilidade` que representa a habilidade em questão e a quantidade de lotes que Marta deverá confeccionar, que seria, neste caso, cinco.

A representação do indivíduo foi feita criando-se a classe `ProcessoIndividual` como demonstra a Figura 11:

A classe `ProcessoIndividual` herda da classe `Individual` do *framework*, e por isso, esta passa a ter um `ArrayList` com objetos do tipo `Chromosome`. Neste caso, como a classe `ProcessoChromosome` herda de `Chromosome` este `ArrayList` terá objetos do tipo `ProcessoChromosome`.

A criação dos cromossomos que irão compor o indivíduo é feita através do construtor da classe `ProcessoIndividual` e, é neste ponto, que os lotes são distribuídos para cada atividade_costureira. O `construtor` da classe `ProcessoIndividual` recebe como parâmetro

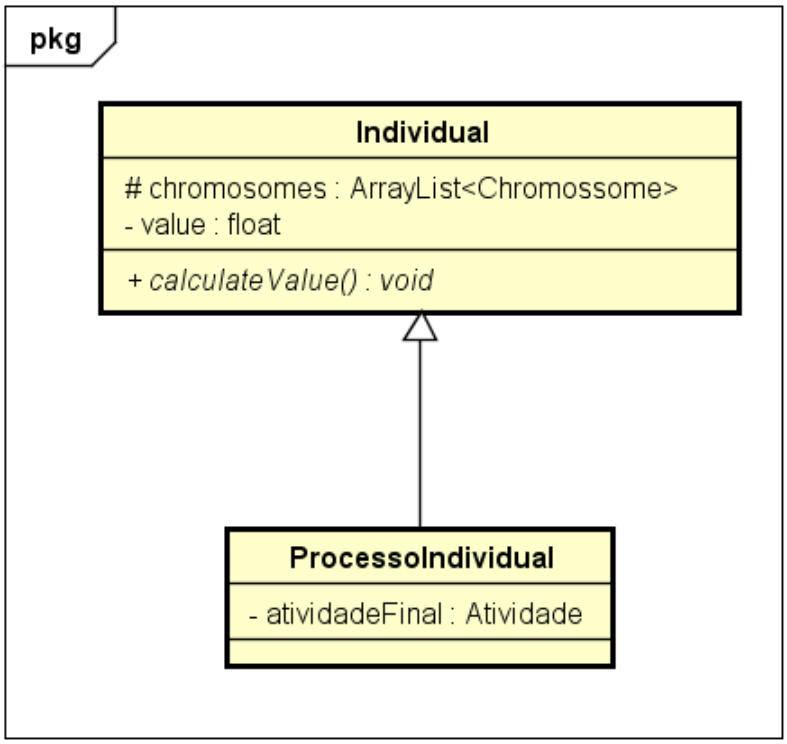

Figura 12 – Classe ProcessoIndividual **Fonte:** Desenvolvido pelos autores.

um objeto representando a atividade final, que será utilizado pela função de avaliação mais adiante, e um `HashMap` que possui como chave o `ID` de uma atividade e uma lista do tipo `CostureiraHabilidade` contendo as costureiras e o tempo gasto por cada uma para fazer tal atividade.

Com base neste `HashMap` então é feita a criação dos cromossomos do indivíduo. Em um primeiro momento, a distribuição de tarefas entre as costureiras seria feita em forma de porcentagem, ou seja, o algoritmo distribuiria uma porcentagem aleatória para cada costureira de uma determinada atividade, desta forma a distribuição seria feita da forma demonstrada na 3.4.

Código 3.4 – Criação de cromossomos (Primeira Abordagem). **Fonte:** Elaborado pelos autores.

```
1
2  package edu.univas.edu.tcc.ga_code;
3
4  import java.util.ArrayList;
5
6  public class ProcessoIndividual extends Individual {
7
8    public Atividade atividadeFinal;
9
10   public ProcessoIndividual(Atividade atividadeInicial,
11     Map<Integer, List<CostureiraHabilidade>> atividadesCostureiras){
12
13     chromosomes = new ArrayList<Chromosome>();
14     this.atividadeFinal = atividadeInicial;
```

```
15
16        for(Integer key: atividadesCostureiras.keySet()){
17           for(CostureiraHabilidade costureira :
18             atividadesCostureiras.get(key)){
19
20             Float porcentagem = (float) (Math.random() *1);
21             chromosomes.add(new ProcessoChromosome(key,
22               costureira, porcentagem));
23           }
24        }
25     }
26  }
```

Feita a distribuição da porcentagem, antes de fazer o cálculo do indivíduo, seria então realizado um cálculo de normalização para que se pudesse encontrar o número de lotes a ser produzido por cada costureira em cada atividade. Tomando como exemplo a Figura 13, este cálculo seria feito da seguinte forma:

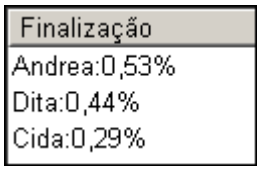

Figura 13 – Distribuição em porcentagem **Fonte:** Desenvolvido pelos autores

- Primeiramente deveria ser feito a soma de todas as porcentagens distribuídas, logo:

  `0,53 + 0,44 + 0,29 = 1,26;`

- o segundo passo seria calcular quanto cada porcentagem equivale dentro do total, neste sentido o cálculo, já fazendo o arredondamento, seria:

  `0,53 / 1,26 = 0,42 | 0,44 / 1,26 = 0,35 | 0,29 / 1,26 = 0,23`

  Logo, neste caso a Andrea seria responsável por 42%, a Dita por 35% e a Cida por 23%;

- Assim, seria feito um cálculo com regra de 3 com o número total de peças. Supondo que o valor total fosse 500, logo:

  `(500 * 42) / 100 = 210 | (500 * 35) / 100 = 175 | (500 * 23) / 100 = 115`

  Neste caso então, a Andrea deveria produzir 210 peças, a Dita 175 e a Cida 115 peças;

- Por fim deveria ser feito uma divisão dos número de peças de cada costureira pela quantidade de peças por lote, que neste caso poderia ser 50, então realizando o cálculo já com arrendondamento:

`210 / 50 = 4 | 175 / 50 = 4 | 115 / 50 = 2`

Assim, a Andrea produziria 4 lotes, a Dita 4 e a Cida 2, dando o total dos 10 lotes a serem produzidos para a atividade de finalização.

Os passos acima para distribuição de atividades seriam então repetidos para cada atividade chave do `HashMap` realizando tal distribuição para cada costureira da lista de costureiras de cada atividade. No momento de fazer o último cálculo, foi feito um arredondamento, com isso se uma costureira tivesse tido uma porcentagem muito pequena, o valor de lotes para esta seria 0, eliminando-a assim da distribuição.

Todavia verificou-se que, distribuindo desta forma, em alguns casos, o total de lotes por atividade não era distribuído de forma correta. Devido ao arredondamento, as vezes uma atividade ficava com lotes a menos ou lotes a mais do que o total definido, o que poderia causar erros no cálculo final. Além disso, no momento de definir como seria o cruzamento surgiu uma questão importante que é o fato de que todas as vezes que fosse criado um indivíduo a partir de outros, deveria ser realizado o cálculo de normalização, e com isso a distribuição de lotes no novo indivíduo poderia ficar completamente diferente de seus pais, resultando assim na quebra do paradigma de algoritmos genéticos que descreve que os indivíduos filhos devem ser formados pela mesclagem das características dos pais.

Buscou-se então uma outra alternativa para se realizar a distribuição dos lotes e definiu-se que, ao invés de distribuir a porcentagem, a distribuição já deveria ser feita a nível de lote sendo esta também realizada de forma aleatória. Os parâmetros do `construtor` da classe `ProcessoIndividuo` permaneceram da mesma forma, alternando somente a forma com que os lotes são distribuídos entre as costureiras em cada atividade conforme mostra o Código 3.5.

Código 3.5 – Distribuição em lotes diretamente. **Fonte:** Elaborado pelos autores.

```
1
2  public ProcessoIndividual(Atividade atividadeFinal,
3    BigDecimal prazoEmSegundos,
4    Map<Integer, List<CostureiraHabilidade>> atividadesCostureiras,
5    int numeroLote, int pecasPorLote){
6
7    chromosomes = new ArrayList<Chromosome>();
8    Map<Integer, ProcessoChromosome> chromossomosMap =
9      new HashMap<Integer, ProcessoChromosome>();
10
11   this.atividadeFinal = atividadeFinal;
12   this.numeroLote = numeroLote;
13   this.pecasPorLote = pecasPorLote;
14   this.prazo = prazoEmSegundos;
15
16   boolean distribuiuPorTodasCostureiras = false;
17
18   int qtdeLote = 0;
```

```
19    int cont = 0;
20
21    for (Integer key : atividadesCostureiras.keySet()) {
22      qtdeLote = numeroLote;
23      cont = 0;
24      int loteCostureira = 0;
25      distribuiuPorTodasCostureiras = false;
26
27      while (true){
28        /* Verificou-se que quando a qtdeLote era 1 o valor
29           sorteado nunca era zero usando o Math.random com
30           CAST para INT */
31        if(qtdeLote == 1){
32          loteCostureira  = Math.round((float) Math.random() * 1);
33        }else{
34          loteCostureira = (int) (Math.random() * qtdeLote);
35        }
36
37        CostureiraHabilidade costureiraHabilidade =
38          atividadesCostureiras.get(key).get(cont);
39
40        ProcessoChromosome intermediario =
41          chromossomosMap.get(costureiraHabilidade.
              getIdCostureiraHabilidade());
42
43        if(intermediario == null){
44          ProcessoChromosome pc = new ProcessoChromosome
45            (key, costureiraHabilidade,loteCostureira);
46
47          chromossomosMap.put
48            (costureiraHabilidade.getIdCostureiraHabilidade(), pc);
49
50          chromosomes.add(pc);
51        }else{
52          int oldValue = intermediario.getQuantidade_lotes();
53          int newValue = oldValue + loteCostureira;
54          intermediario.setQuantidade_lotes(newValue);
55          intermediario.setLotesToShow(newValue);
56        }
57
58        if (cont == atividadesCostureiras.get(key).size() - 1) {
59          cont = -1;
60          distribuiuPorTodasCostureiras = true;
61        }
62
63        qtdeLote -= loteCostureira;
64
65        if(qtdeLote == 0 && distribuiuPorTodasCostureiras){
66          break;
67        }
68        cont++;
69      }
70    }
71  }
```

Conforme descrito no Código 2.3, é feita uma iteração no `HashMap` e, para cada atividade, é feita a distribuição dos lotes para as costureiras desta lista. O algoritmo então atribui a cada costureira um valor que pode variar de 0 até `qtdeLote`. Assim é criado objetos da classe `ProcessoChromosome` e colocado na lista de cromossomos do indivíduo. Após a criação de

cada cromossomo, a quantidade de lotes é subtraída pelo valor atribuído ao cromossomo recém criado. Se a iteração passar por todas as costureiras da lista, o contador é reiniciado e então, se ao final sobrar lotes a serem distribuídos, a distribuição recomeça na primeira costureira, acrescentando assim seu número de lotes de acordo com o novo valor sorteado. A iteração termina quando não há mais lotes a serem distribuídos e a execução já passou por todas as costureiras. Em uma primeira versão, quando a execução chegava ao final da lista de costureiras e ainda existiam lotes a serem distribuídos, este restante de lotes era atribuído à última costureira, porém verificou-se, que desta forma, o algoritmo tendia a nunca distribuir zero lotes à última costureira, o que causou resultados ineficazes na distribuição. Alterando para esta forma, a distribuição passou a ser feita de maneira totalmente uniforme deixando o resultado coerente. Como é possível perceber, neste processo uma costureira pode receber aleatoriamente o valor 0, o que irá resultar na sua eliminação do processo da mesma forma que iria ocorrer na primeira abordagem. Por fim, após a finalização do primeiro `for` um novo indivíduo terá sido criado, semelhante ao quadro apresentado na Figura 7.

Concluindo, a distribuição das atividades ocorre todas as vezes que se cria um novo indivíduo. Tais indivíduos podem ser criados no processo de criação da `população inicial`, na criação de `indivíduos estrangeiros` e no processo de `cruzamento`, como será descrito nas próximas seções, ressaltando porém que no processo de cruzamento, os cromossomos do indivíduo é a mistura dos cromossomos dos pais, já criados anteriormente, e portanto há também um construtor na classe `ProcessoIndividual` que recebe uma lista de cromossomos para se criar um novo indivíduo. Este processo será demonstrado na seção que descreve o cruzamento.

3.4.4 Função de avaliação

Após a criação da população inicial, esta é então submetida a um processo de avaliação. Assim é feita uma iteração sobre a lista de indivíduos e para cada um é chamado então o seu método `calculateValue()`. Tal método é declarado de forma abstrata na classe mãe `Individual` e implementado na classe `ProcessoIndividual`.

Assim como já foi visto anteriormente, cada costureira sabe fazer uma ou mais partes da calça e gasta um determinado tempo, medido em segundos, além de cobrar um valor para se produzir cada peça, que varia de acordo com a habilidade, além disto, existe um tempo de transporte entre cada costureira que é medido através da distância euclidiana, ou seja, cada costureira recebe um valor X e Y que representam suas localidades e então, através da formula

euclidiana, é possível calcular a distância entre elas. A Figura 14 demonstra os dados a serem considerados para o cálculo do valor do indivíduo.

Habilidade	Costureira	Tempo/Peça	Preço/Peça	Posição X	Posição Y
Finalização	Josi	3s	R$ 1,50	30	10
Carimbo	Marcelo	0s	R$ 0,00	10	20
Frente	Roberta	4s	R$ 1,40	20	30
Frente	Tereza	2s	R$ 2,50	15	35
Frente	Maria	3s	R$ 1,70	25	43

Figura 14 – Demonstração de costureiras e habilidades **Fonte:** Desenvolvido pelos autores.

Com base nestes dados é realizado um cálculo a fim de se encontrar o tempo total de fabricação do número de peças desejado e, ao fim, um valor de tempo e custo de produção será atribuído ao indivíduo, tais valores serão utilizados posteriormente no processo de classificação.

Para o desenvolvimento do cálculo do tempo de produção, foi necessário construir uma estrutura para representar a questão da ordem de precedência entre as atividades. Tal estrutura, conforme é demonstrado na Figura 15, deveria ter nós representando cada atividade, as costureiras que trabalham em cada atividade e o número de lotes atribuídos a cada uma aleatoriamente pelo algoritmo.

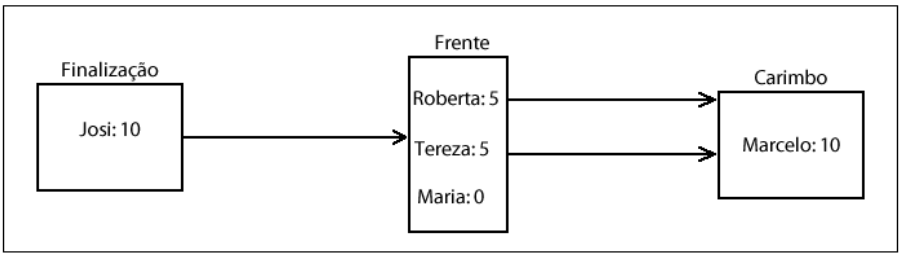

Figura 15 – Estrutura de representação da ordem de precedência **Fonte:** Desenvolvido pelos autores.

Como foi visto na seção 1.4.2, cada indivíduo possui uma lista de cromossomos, e cada cromossomo, por sua vez, representa a alocação de uma costureira, contendo os lotes que esta deve produzir para uma determinada atividade. Desta forma, para calcular o custo total de produção, foi realizada uma iteração na lista de cromossomos do indivíduo somando o preço total de confecção dos lotes que cada costureira recebeu aleatoriamente pelo algoritmo conforme mostra o Código 3.6.

Para o cálculo do tempo de produção, definiu-se então que tal lista de cromossomos deveria ser dividida de forma que se pudesse agrupar os cromossomos por atividade estabelecendo assim a relação demonstrada na Figura 16, para, que por fim, o cálculo pudesse ser realizado.

Para isso, conforme demonstrado no Código 3.6, primeiramente, a lista de cromossomos foi distribuída em um `HashMap` denominado `atividadeCromossomos`, contendo como chave a atividade e como valor a lista de cromossomos para a respectiva atividade e foi criado uma classe denominada `Node`, sendo esta a responsável por criar a estrutura mostrada na Figura 16.

O método `calculateValue()`, após agrupar os cromossomos por atividade, instancia um objeto da classe `Node` passando a `atividadeFinal` e o número de peças por lote recebidos na criação do indivíduo, conforme descrito na seção anterior, e o map atividadeCromossomos conforme mostra a Código 3.6.

Código 3.6 – Método `calculateValue()`. **Fonte:** Elaborado pelos autores.

```
1
2   public void calculateValue() {
3     Map<Integer, List<Chromosome>> atividadeCromossomos
4       = new HashMap<Integer, List<Chromosome>>();
5
6     Integer lastAtividade = null;
7     float custoTotal = 0;
8     int totalPecasAProduzir = 0;
9     List<Chromosome> cromossomos = null;
10
11    /*Calcular o custo total*/
12    for(Chromosome chromosomeCusto : chromosomes){
13      totalPecasAProduzir = 0;
14      ProcessoChromosome processoChromossome =
15        (ProcessoChromosome) chromosomeCusto;
16
17      totalPecasAProduzir =
18        processoChromossome.getLotesToShow() * this.pecasPorLote;
19
20      custoTotal += totalPecasAProduzir *
21        processoChromossome.getCostureiraHabilidade()
22          .getPrecoPorPeca();
23    }
24
25    for (Chromosome chromosome : chromosomes) {
26    ProcessoChromosome processoChromossome = (ProcessoChromosome)
          chromosome;
27
28      if (lastAtividade == null || lastAtividade !=
            processoChromossome.getAtividade()) {
29        cromossomos = new ArrayList<Chromosome>();
30
31        atividadeCromossomos.put
32          (processoChromossome.getAtividade(),cromossomos);
33        lastAtividade = processoChromossome.getAtividade();
34      }
35      cromossomos.add(processoChromossome);
36    }
37    node = new Node(atividadeFinal, atividadeCromossomos,this.
          pecasPorLote);
38
39    setCusto(custoTotal);
40
41
42    /*So deve-se calcular o valor do individuo se
```

```
43        ele nao foi calculado ainda porque uma vez calculado
44        o valor o numero de lotes no objeto cromossomo foi
45        decrementado */
46      if (getValue() == 0) {
47        setValue(node.getValorTotal());
48        setRootNode(node);
49      }
50    }
```

A estrutura da classe `Node` foi realizada de forma a produzir objetos de si mesma de forma recursiva, assim cada vez que esta é instanciada, é como se criasse um nó daqueles mostrados na Figura 16.Assim, quando o método `calculateValue()` instancia um objeto `Node`, no construtor deste outros nós são criados, e então toda estrutura, como foi ilustrada na Figura 16, será criada. O Código 2.6 mostra a construção de um objeto `Node`.

Código 3.7 – Construtor da classe Node. **Fonte:** Elaborado pelos autores.

```
1
2   public Node(Atividade atividade,Map<Integer,List<Chromosome>>
3      atividadeCromossomos, int pecasPorLote){
4
5     this.atividade = atividade;
6     this.pecasPorLote = pecasPorLote;
7     cromossomos = atividadeCromossomos.
8       get(atividade.getIdAtividade());
9
10    Atividade atividadePredecessora = null;
11
12    for(AtividadeOrdem predecessora :
13      atividade.getAtividadeOrdemsForIdAtividade()){
14
15      atividadePredecessora =
16        predecessora.getAtividadePredecessora();
17
18      predecesoras.add(new Node(atividadePredecessora,
19        atividadeCromossomos,pecasPorLote));
20    }
21  }
```

A classe `Node` recebe em seu construtor um objeto de Atividade, que na primeira vez que o objeto for instanciado será a atividade final, o `MAP` com todos os cromossomos e o número de peças por lote, que será utilizado posteriormente. O construtor então armazena as informações e pega do `MAP` somente os cromossomos relacionados a atividade recebida e, por fim, faz uma iteração na lista de atividades predecessoras da atividade recebida e, recursivamente, cria novos nós, construindo assim, a estrutura demonstrada na Figura 16.

Como se pode ver no método `calculateValue()` no Código 2.5, após criar a estrutura de nós, é chamado o método `getValorTotal()` do objeto `node` criado. Este método é responsável por iniciar a sequência lógica que faz o cálculo do tempo total a ser gasto pelo indivíduo, calculando o tempo gasto por cada costureira, definindo quem irá enviar peças pra

quem e calculando o tempo de transporte de cada envio, conforme demonstra Código 3.8 .

Código 3.8 – Método `getValorTotal()`. **Fonte:** Elaborado pelos autores.

```
1
2   public long getValorTotal(){
3     long valor =  0;
4     long maior = 0;
5
6     for(Chromosome chromosome : cromossomos){
7       ProcessoChromosome processoChromosome =
8         (ProcessoChromosome) chromosome;
9
10      if(processoChromosome.getQuantidade_lotes() > 0){
11        valor = getCromossomeValue(processoChromosome,
12          processoChromosome.getQuantidade_lotes());
13      }
14
15      if(valor > maior){
16        maior = valor;
17      }
18    }
19    return maior;
20  }
```

O método faz uma iteração na lista de cromossomos do nó da atividade final e irá chamar o método getChromosomeValue passando cada cromossomo e o valor de seus lotes, e irá retornar o valor do maior cromossomo.

Tomando como base a Figura 17, para facilitar o entendimento, o método getChromosomeValue() será chamado passando o cromossomo "Josi"e o inteiro 10 na quantidade de lotes. Este método é responsável por calcular o tempo gasto pela costureira para realizar os lotes atribuídos a ela. O tempo gasto pela costureira é definido por `NLC * QPL * TP`, onde: `NLC` é o número de lotes atribuídos para a costureira, `QPL` é a quantidade de peças por lote e o `TP` é o tempo que a costureira gasta para fazer cada peça, porém este tempo também é influenciado pelo tempo que se é gasto para receber as partes dependentes, conforme demonstra o código 3.9.

Código 3.9 – Método `getCromossomeValue()`. **Fonte:** Elaborado pelos autores.

```
1
2 public long getCromossomeValue(ProcessoChromosome processoChromosome,
3   int qtdeLote){
4
5   long valor = 0;
6   valor = qtdeLote * this.pecasPorLote *
7      processoChromosome.getCostureiraHabilidade().getTempoPorPeca();
8
9   valor += getTempoRecebimentoPecas(processoChromosome,qtdeLote);
10  return valor;
11 }
```

O método `getChromossomeValue()` chama então o método getTempoRecebimentoPecas(), passando o cromossomo Josi e o inteiro 10 como quantidade de lote. O método chamado tem a função de fazer uma busca nos nós predecessores buscando encontrar qual o tempo gasto para o recebimento das partes predecessoras da atividade e retornar o maior valor, conforme demonstra o Código 3.10.

Código 3.10 – Método `getTempoRecebimentoPecas()`. **Fonte:** Elaborado pelos autores.

```
1
2   public long getTempoRecebimentoPecas(
3     ProcessoChromosome processoChromosome, int qtdeLote){
4
5     long valor = 0;
6     long maior = 0;
7
8     for(Node node : predecesoras){
9       valor = node.getValueChromosomosPredecessores(
10        processoChromosome, qtdeLote);
11
12      if(valor > maior){
13        maior = valor;
14      }
15    }
16    return maior;
17  }
18 }
```

Neste ponto começa então um processo recursivo, pois é chamado um método da própria classe Node, só que de uma outra instância. O método chamado é o getValueChromosomosPredecessores() passando o cromossomo Josi e o inteiro 10, como número de lotes. O código 3.11 mostra este método.

Código 3.11 – Método `getValueChromosomosPredecessores()`. **Fonte:** Elaborado pelos autores.

```
1
2   public long getValueChromosomosPredecessores(
3     ProcessoChromosome processoChromosome,int qtdeLote){
4
5     int qtdeEachCromossome = 0;
6     long valor = 0;
7     long maior = 0;
8     long distance = 0;
9
10    for(Chromosome chromosome : cromossomos){
11      ProcessoChromosome processoChromosomeBefore =
12         (ProcessoChromosome) chromosome;
13
14      qtdeEachCromossome =
15         processoChromosomeBefore.getQtdeLotes(qtdeLote);
16
17      if(qtdeEachCromossome > 0){
18        qtdeLote -= qtdeEachCromossome;
19        valor = getCromossomeValue(
20          processoChromosomeBefore, qtdeEachCromossome);
21
22        distance = calcularTempoEntreCostureiras(
```

```
23            processoChromosome, processoChromosomeBefore);
24
25          processoChromosome.addCostureiraPredecessora(
26            processoChromosomeBefore.getCostureiraHabilidade(),
27
28          qtdeEachCromossome,distance,valor);
29
30          valor += distance;
31        }
32
33        if(valor > maior){
34          maior = valor;
35        }
36
37        if(qtdeLote == 0){
38          break;
39        }
40      }
41      return maior;
42    }
```

Tal método é responsável por iterar sobre a lista de cromossomos do nó anterior buscando de qual ou quais costureiras podem ser obtidos os lotes, retornando o maior valor. No exemplo da Figura 16, a atividade anterior é a "Frente" e a primeira costureira da lista é a Roberta, neste caso a Josi solicita 10 lotes da parte da frente para a Roberta, porém, neste caso, a Roberta confeccionou somente 5 lotes, neste caso , a Josi irá consumir os 5 lotes confeccionados.

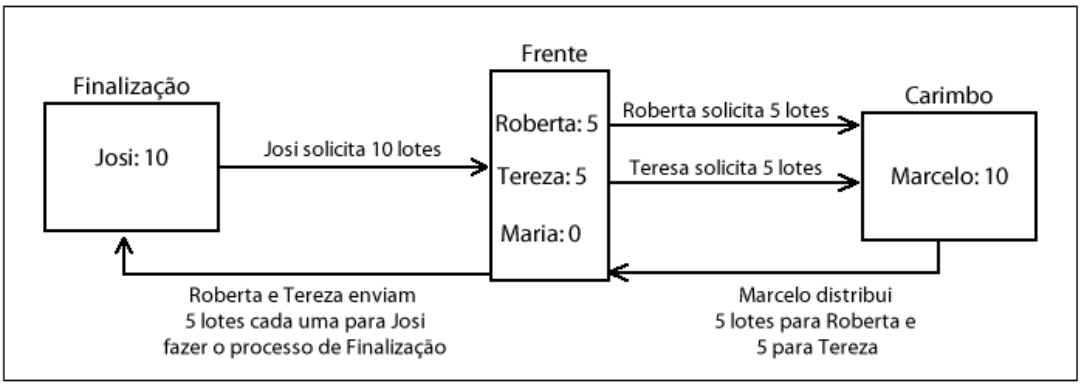

Figura 16 – Exemplo de solicitação de lotes predecessores **Fonte:** Desenvolvido pelos autores.

Assim, a costureira Roberta é adicionada ao cromossomo Josi como costureira predecessora e novamente é chamado o método `getChromosomeValue()`, só que agora passando o cromossomo Roberta e a quantidade de lote que ela deve produzir para atender a Josi, para que se possa calcular o tempo, além disso será chamado o método calcularTempoEntreCostureiras() que irá retornar o tempo de transporte entre a Josi e a Roberta, e será somado ao valor retornado de `getChromosomeValue()` que foi chamado passando o cromossomo Roberta. O Código 3.12 demonstra este método `calcularTempoEntreCostureiras()`.

Código 3.12 – Método `getValueChromosomosPredecessores()`. **Fonte:** Elaborado pelos autores.

```
1
2  public long calcularTempoEntreCostureiras(
3    ProcessoChromosome processoChromosome,
4    ProcessoChromosome processoChromosomeBefore){
5
6    int posicaoCostureiraX = processoChromosome.
7      getCostureiraHabilidade().
8      getCostureira().getPositionX();
9
10   int posicaoCostureiraY = processoChromosome.
11     getCostureiraHabilidade().
12     getCostureira().getPositionY();
13
14   int posicaoCostureiraBeforeX = processoChromosomeBefore.
15     getCostureiraHabilidade().
16     getCostureira().getPositionX();
17
18   int posicaoCostureiraBeforeY = processoChromosomeBefore.
19     getCostureiraHabilidade().
20     getCostureira().getPositionY();
21
22   long distance = (long) Math.sqrt(
23   Math.pow(posicaoCostureiraX - posicaoCostureiraBeforeX, 2)+
24   Math.pow(posicaoCostureiraY - posicaoCostureiraBeforeY, 2));
25
26   return distance * 100;
27 }
```

Seguindo as execuções dos métodos já explicados, o fluxo de chamadas então será `getCromossomeValue()`, `getTempoRecebimentoPecas()`, getValueChromosomosPredecessores() e neste ponto o cromossomo Roberta irá solicitar para sua atividade anterior, 5 lotes, conforme mostra a Figura 16.

Neste caso, a atividade anterior é o Carimbo, neste ponto existe uma exceção, pois quando se solicita para atividade Carimbo, não são consumidos os lotes desta, pois, conforme definido no escopo explicado anteriormente, a atividade de Carimbo só possui o Marcelo como trabalhador e ele apenas distribui o material de costura, assim qualquer solicitação feita a ele é correspondida, além disso como o tempo de produção e o custo por peça do Marcelo é zero, só sera considerado apenas o tempo de transporte entre o Marcelo e a Roberta.

E assim, todo o processo é feito recursivamente, na qual as costureiras da primeira atividade vão consumindo os lotes das costureiras das atividades predecessoras, conforme mostra a Figura 16. Como a Roberta não conseguiu atender a Josi, uma vez calculado o tempo de produção ds 5 peças da Roberta, o algoritmo verifica a próxima costureira da atividade Frente, que seria a Tereza neste exemplo, e, da mesma forma, calcula o tempo de produção do número de peças solicitadas que seria 5 neste caso.

Concluindo, o processo é todo feito recursivamente, na qual as costureiras das primeiras atividades vão consumindo os lotes das costureiras das atividades predecessoras e somado o

tempo de produção e o tempo de transporte entre elas, conforme mostra a Figura 17.

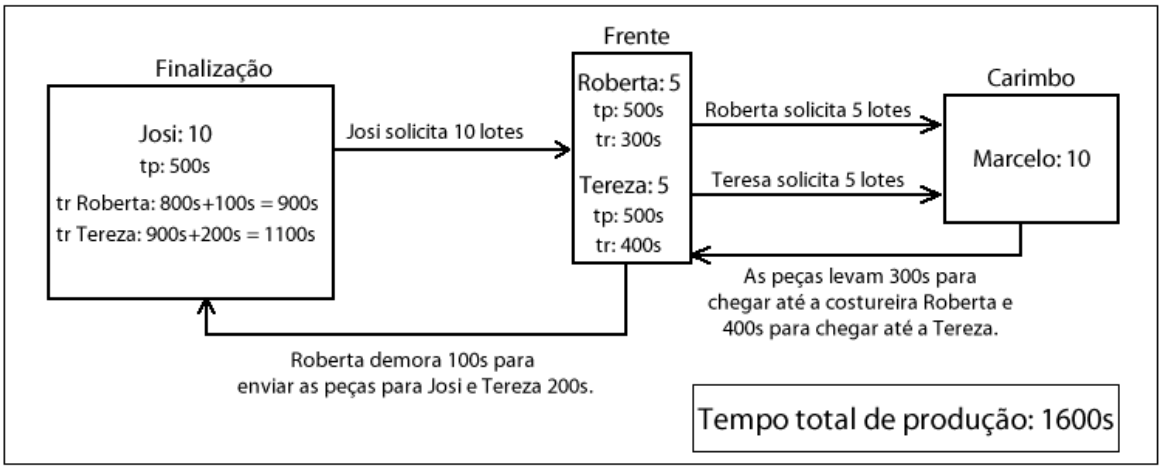

Figura 17 – Distribuição demonstrando o tempo **Fonte:** Desenvolvido pelos autores.

A recursividade para quando se chega na atividade Carimbo, então os valores começam a ser retornados. No fim, prevalece sempre o maior valor e assim o método `getValorTotal()` irá retornar o tempo total de produção das partes e o transporte das mesmas entre as costureiras e então este valor é colocado no atributo `value` do indivíduo.

3.4.5 Classificação dos indivíduos

3.4.6 Criação de uma nova população

Primeiro falar de indivíduos estrangeiros, seleção, cruzamento, mutação e atualização da população

O método `execute()` da classe `GAController` é responsável por coordenar toda execução do algoritmo. AQUI TENHO QUE EXPLICAR TODA A EXECUÇÃO DO ALGORITMO PASSANDO PELA SELEÇÃO, CRUZAMENTO, MUTAÇÃO E INDIVÍDUOS ESTRANGEIROS.

O conceito de indivíduos estrangeiros considera o fato de que, na natureza, durante o processo de geração de uma nova população, indivíduos estrangeiros podem começar a fazer parte de tal população. No algoritmo genético, tais indivíduos simplesmente são introduzidos à nova população de acordo com uma taxa definida no atributo `foreignIndividualRate` da classe GAModel do *framework*, conforme já explicado anteriormente, e os indivíduos são criados seguindo a mesma lógica utilizada para a criação da população inicial, a diferença é que, ao criar tais indivíduos, o *looping* é executado de acordo com a taxa definida.

3.4.7 Construção da interface gráfica, CRUD e apresentação dos dados

Este item já foi implementado porém ainda não documentado.

4 DISCUSSÃO DE RESULTADOS

Neste capítulo serão apresentados e discutidos os resultados obtidos pela pesquisa e implementação do sistema de otimização do processo de fabricação de calças. Tal discussão será realizada em forma de casos de teste, buscando demonstrar o comportamento do algoritmo genético de diferentes formas.

Desde o princípio, quando começamos a discutir sobre o tema do nosso trabalho de conclusão de curso, tínhamos em mente desenvolver algo relacionado à inteligência artificial, por ser um assunto de bastante relevância na área de desenvolvimento de software. Dentro deste campo então, realizando algumas buscas na internet, encontramos o assunto de algoritmos genéticos. Coincidentemente no primeiro semestre deste ano, tivemos aulas de sistemas especialistas a qual o assunto foi abordado o que facilitou bastante o aprendizado.

Assim, como sugestão do professor orientador, decidimos então desenvolver uma aplicação para otimizar um processo de distribuição de atividades entre costureiras para um microempresário da cidade de Cachoeira de Minas - MG, do ramo de costura e constatamos que um sistema Web seria mais cômodo de ser utilizado por não precisar de nenhuma instalação e o usuário poder acessar o programa de qualquer lugar desde que estivesse conectado à internet. Neste sentido, como já estávamos familiarizados com JSF e Primefaces. decidimos adotá-los como tecnologias para o desenvolvimento em plataforma Web, além disso usamos um *plug-in* denominado `JBoss Tool` o qual foi de grande utilidade para gerar as classes modelo a partir do banco de dados.

Inicialmente tomamos como base o caso do microempresário citado acima, todavia logo após iniciarmos o trabalho, o mesmo fez uma reestruturação de processos em sua empresa. Desta forma, sua maneira de trabalhar deixou de ser um cenário o qual algoritmos genéticos pudessem ser aplicados, desta forma, decidimos fechar um escopo para que o trabalho pudesse continuar, conforme descrito na seção reuniões do quadro metodológico. Com o escopo definido, o foco passou a ser na definição da estrutura dos elementos do algoritmo genético.

Como já explanado no quadro teórico, segundo Linden (2012), a estrutura do algoritmo genético é composta por populações que são formadas por indivíduos que por sua vez são formados por cromossomos. Cada indivíduo representa uma solução e cada cromossomo do indivíduo representa uma de suas características. Assim então é gerado uma população inicial de indivíduos e, a partir desta, um processo de cruzamento e mutação é iniciado a fim de que possa ser gerados novos indivíduos que representem soluções ainda melhores que seus antecessores.

Uma das maiores dificuldades nesta etapa, foi definir como os indivíduos e cromossomos seriam estruturados. Assim em uma discussão com o professor Artur Barbosa, decidimos que o total de peças deveriam ser divididos em lotes e que cada cromossomo deveria ter uma costureira e a quantidade de lotes a ser confeccionado por esta, Assim o indivíduo teria um conjunto de cromossomos que representaria quais costureiras e o quanto elas iriam trabalhar naquele processo, representando uma solução para a distribuição.

Com base nesta definição, um dos papeis desempenhados pelo algoritmo, seria a distribuição aleatória do trabalho. Neste sentido, nos deparamos com uma outra dificuldade que foi a de definir a população inicial de indivíduos, ou seja, como os lotes de peças seriam distribuídos para cada cromossomo. Inicialmente pensamos em distribuir porcentagens para cada costureira e posteriormente fazer um cálculo de normalização, conforme explica a sessão Distribuição das atividades do quadro metodológico porém, constatamos que o cálculo não era preciso e que, ao cruzar dois indivíduos, os indivíduos filhos, após o cálculo de normalização, ficavam completamente diferente de seus progenitores fugindo assim da ideia de algoritmos genéticos. Definimos então distribuir as atividades já nível de lote o que resolveu a questão.

Um dos maiores desafios do trabalho foi a definição da função de avaliação, pois, uma vez que existe uma estrutura de nós contendo as atividades e o cálculo do tempo de cada atividade depende de nós predecessores, a maior parte da função foi pensada para ser desenvolvida de forma recursiva o que dificultou o `debug`.

Para a construção do algoritmo genético, foi utilizado uma base desenvolvida pelo professor Artur Barbosa durante as aulas de sistemas especialistas, que define uma série de regras a ser seguida durante o desenvolvimento. Tal base é explicada com mais detalhes no quadro metodológico e foi de grande ajuda pois, além de definir tais regras de desenvolvimento a base já implementava os métodos de seleção e cruzamento, neste sentido só tivemos que realizar algumas adaptações nestes para que pudessem se adequar à nossa lógica.

Após ter o escopo definido e a modelagem do algoritmo genético realizada, dividimos o desenvolvimento entre os autores e mantivemos reuniões periódicas para acertar detalhes do desenvolvimento.

Após todas as etapas descritas acima, obtivemos como resultado a aplicação capaz de distribuir atividades de forma a se obter o menor custo e o menor tempo de produção, alcançando assim nossos objetivos específicos.

Com a aplicação finalizada, foram realizados então casos de testes a fim de colocar em prova a eficiência da ferramenta para se buscar melhores soluções nas distribuição de tarefas, conforme mostra as seções seguintes.

4.1 Teste com somente uma atividade

Este teste foi realizado com apenas uma atividade, neste caso não está sendo considerado que as costureiras devem ir até a fábrica para retirar os tecidos com o Marcelo. Para este teste foi cadastrado então um processo, criando-se apenas uma atividade para este, conforme mostra a Figura 27:

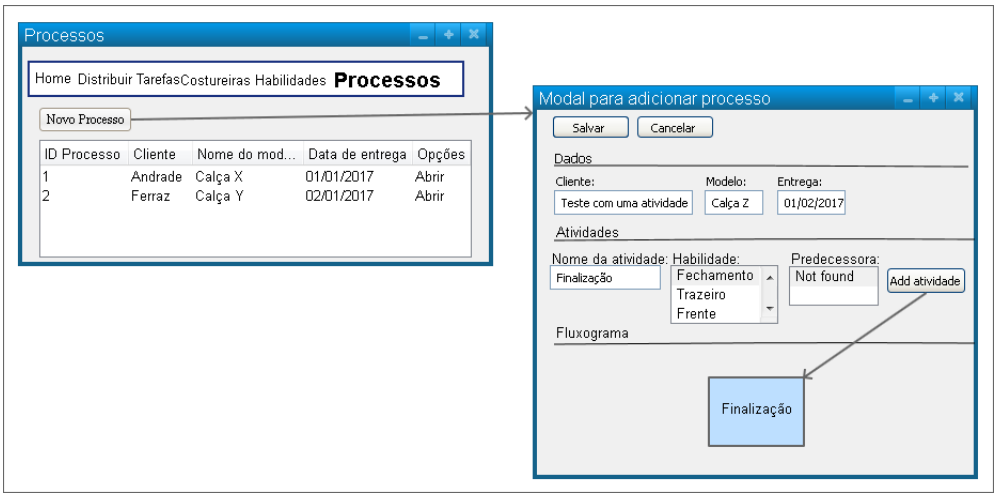

Figura 18 – Caso de teste com uma atividade **Fonte:** Desenvolvido pelos autores

Para a atividade de finalização, atribuída ao processo na Figura 27, foram cadastradas 5 costureiras e seus respectivos tempos para fazer tal atividade, conforme mostra a Figura 28:

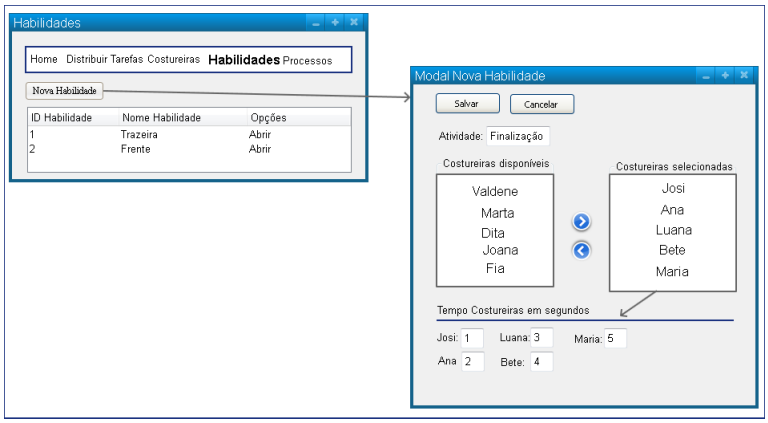

Figura 19 – Cadastro da habilidade e associação das costureiras **Fonte:** Desenvolvido pelos autores

Feito isso foi iniciado então o processo de distribuição das atividades, através do menu Distribuição de Tarefas. O algoritmo foi configurado para ter 1000 gerações e 80 indivíduos em cada população, além disso a taxa de indivíduos estrangeiros e de mutação foram de 0,5% e 0,05% respectivamente, conforme mostra a Figura 29.

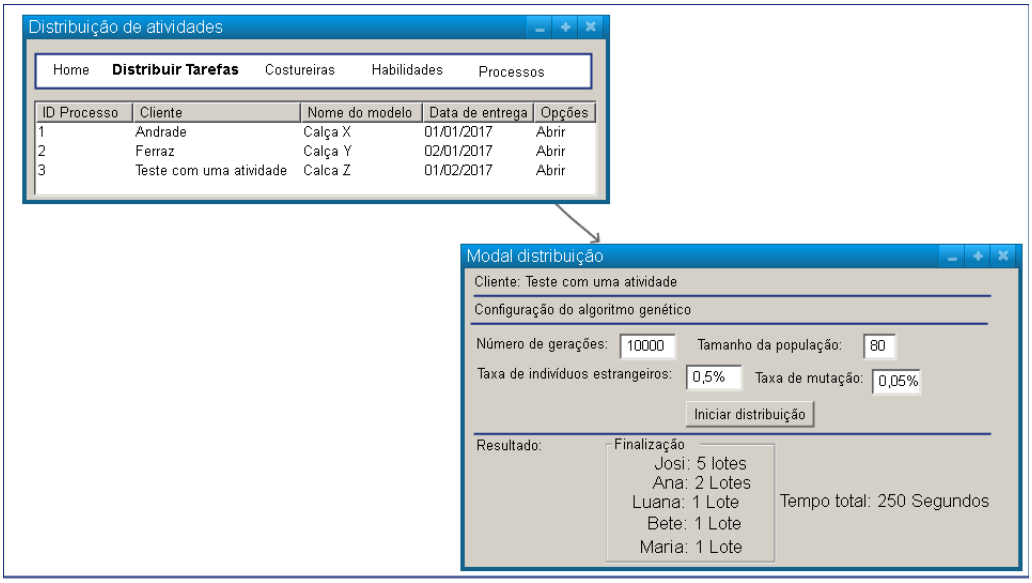

Figura 20 – Distribuição das atividades **Fonte:** Desenvolvido pelos autores

Após clicar no botão "Iniciar Distribuição", o algoritmo distribuiu os lotes para as costureiras da finalização, conforme mostra a Figura 29. O tempo total encontrado foi de 250 segundos. Este é o tempo da costureira que demorou mais tempo para produzir seus lotes. Na Figura 30, pode-se ver o tempo gasto por cada costureira, de acordo com seu tempo de produção para cada peça.

Costureira	Qtde Lote	Peças p/ lote	Total peças	Tempo p/ peça	Total
Josi	5	50	250	1	250 segundos
Ana	2	50	100	2	200 segundos
Luana	1	50	50	3	150 segundos
Bete	1	50	50	4	200 segundos
Maria	1	50	50	5	250 segundos

Figura 21 – Resultado da distribuição **Fonte:** Desenvolvido pelos autores

Baseando-se na Figura 30, pode-se observar que o algoritmo distribui o máximo de peças possível para a Josi, porque ela tem o menor tempo. Nota-se que 5 é o máximo de lotes que a Josi pode receber porque se ela recebesse mais um, seu tempo total passaria a ser 300 segundos, assim o tempo total não seria de 250 segundos. Assim, o algoritmo distribuiu as atividades as outras costureiras de forma uniforme, porém, de forma inteligente, definiu menos lotes para aquelas que gastam mais tempo para se produzir, ou seja, encontrou a melhor forma de distribuir as tarefas para este caso provando assim sua eficiência.

4.2 Teste considerando o tempo de transporte

Este teste foi realizado com duas atividades em que as costureiras estão em suas casas. Neste caso deve-se considerar o tempo de transporte entre as costureiras da atividade Frente e a fábrica do Marcelo para a retirada dos materiais e o tempo entre tais costureiras e a costureira responsável pela finalização, conforme mostra a Figura 31.

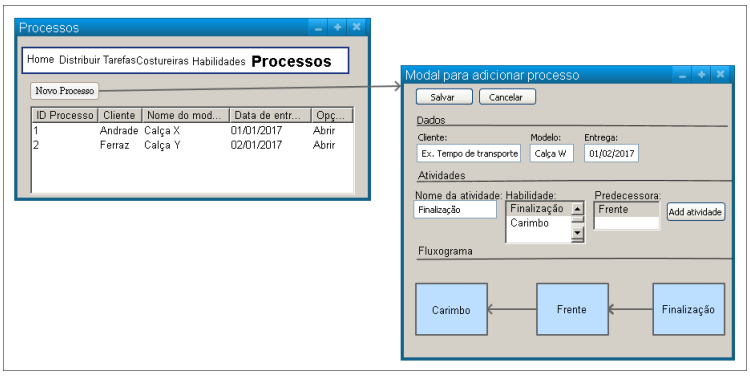

Figura 22 – Caso de teste com tempo de distribuição **Fonte:** Desenvolvido pelos autores

Neste teste foi considerado apenas o tempo de transporte, logo o tempo de confecção por peça das costureiras da parte da Frente possuem o mesmo valor, conforme mostra a Figura 32.

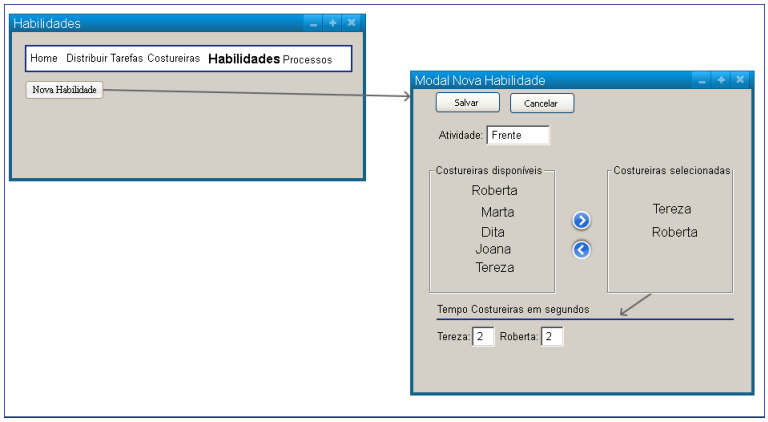

Figura 23 – Caso de teste com tempo de distribuição **Fonte:** Desenvolvido pelos autores

Além disso foi necessário cadastrar um valor X e Y para cada costureira que representa sua localização conforme mostra a Figura 33.

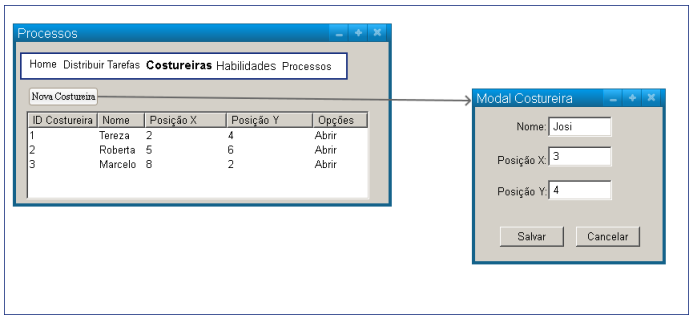

Figura 24 – Cadastro de costureiras **Fonte:** Desenvolvido pelos autores

Feito isso foi iniciado então o processo de distribuição das atividades, através do menu Distribuição de Tarefas. O algoritmo foi configurado para ter 1000 gerações e 80 indivíduos em cada população, além disso a taxa de indivíduos estrangeiros e de mutação foram de 0,5% e 0,05% respectivamente, conforme mostra a Figura 34.

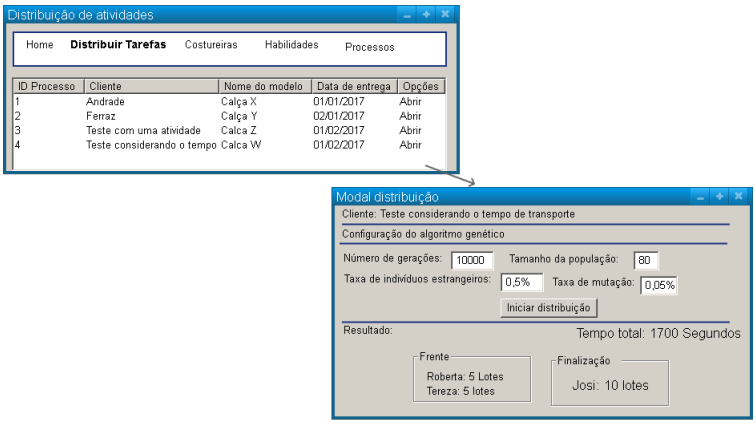

Figura 25 – Distribuição das atividades **Fonte:** Desenvolvido pelos autores

Após clicar no botão "Iniciar Distribuição", o algoritmo distribuiu os lotes para as costureiras das atividades finalização e frente o qual o tempo total encontrado foi de 1700 segundos, conforme mostra a Figura 34. Neste caso o algoritmo distribuiu 5 peças para cada costureira da parte da frente, logo, como a atividade de finalização possui somente uma costureira, a Roberta e a Tereza irão passar 5 peças cada uma para a Josi, que faz a finalização. Conforme descrito no quadro metodológico, o cálculo da distância entre as costureiras é realizado através da fórmula euclidiana multiplicando-se o resultado por 100. A Figura 35 mostra as distâncias entre as costureiras calculadas através desta fórmula com suas posições X e Y.

Costureira 1	Posicao X	Posicao Y	Costureira 2	Posicao X	Posicao Y	Tempo transporte
Tereza	2	4	Josi	3	4	100 segundos
Roberta	5	6	Josi	3	4	200 segundos
Tereza	2	4	Marcelo	8	2	600 segundos
Roberta	5	6	Marcelo	8	2	500 segundos

Figura 26 – Distribuição do tempo **Fonte:** Desenvolvido pelos autores

Considerando que as costureiras Roberta e Tereza gastam o mesmo tempo para fazer a frente e que o tempo de transporte entre a Tereza e a Josi é menor (100 a menos que entre a Roberta e a Josi), o algoritmo deveria distribuir mais peças a Tereza, porém os materiais devem ser pegos na casa do Marcelo e, neste caso, o tempo de transporte entre a Tereza e o Marcelo é 100 a mais que o tempo de transporte entre a Roberta e o Marcelo, logo o algoritmo entendeu que o tempo se igualou e definiu o mesmo número de peças para cada costureira.

Existem outros resultados a serem documentados incluindo a questão de custo, porém até a entrega da discussão dos resultados os mesmos não estavam elaborados.

5 CONCLUSÃO

A conclusão deste trabalho é . . .

Assim conclui-se que. . .

REFERÊNCIAS

BERGSTEN, H. ***Javaserver Faces***. [S.l.: s.n.], 2004.

CARVALHO, L. *Aprenda algumas técnicas de reunião*. 2012. <http://www.administradores.com.br/artigos/negocios/aprenda-algumas-tecnicas-de-reuniao/62516/>. Acessado em 04 de abril.

DATE, C. J. *Introdução a sistemas de banco de dados*. 8º. ed. São Paulo: Campus, 2004.

DOUGLAS, K.; DOUGLAS, S. *PostgreSQL*: A comprehensive guide to building, programming, and administering postgresql databases. 1º. ed. EUA: Sams Publishing, 2003.

FARIA, J. de. *TCC I*. Pouso Alegre: Univás. Notas de Aula 26 de março: [s.n.], 2015.

FARIA, T. *Java EE 7 com JSF, PrimeFaces e CDI*. [S.l.: s.n.], 2013.

FERNANDES, A. M. D. R. *Inteligência Artificial: Noções Gerais.* [S.l.]: Visual Books, 2003.

FILITTO, D. Algoritimos genéticos: Uma visão explanatória. *Saber Acadêmico*, n. 06, p. 136–143, 2008.

FREITAS, C. C. et al. Uma ferramenta baseada em algoritmos genéticos para a geração de tabela de horário escolar. *SÉTIMA ESCOLA REGIONAL DE COMPUTAÇÃO Bahia-Sergipe. Vitória da Conquista:[sn]*, 2007.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.: s.n.], 2009.

GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de Pesquisa.* 1º. ed. Porto Alegre: UFRGS, 2009.

GIL, A. C. *Métodos e técnicas de pesquisa social.* São Paulo: Atlas, 1999.

GOLDBERG, D. E. *Genetic Algorithms in search optimization and machine learning*. 1º. ed. New York: Addison-Wesley publishing company,inc., 1989.

HAGUETTE, T. M. F. *Metodologias qualitativas na Sociologia*. 5º. ed. Petrópolis: Vozes, 1997.

JUNEAU, J. *Primefaces in the Enterprise*. 2014. <http://www.oracle.com/technetwork/articles/java/java-primefaces-2191907.html>. Acesso em: 15 de janeiro de 2015.

JUNIOR, P. J. *P. de JAVA: Guia do Programador.* 1º. ed. [S.l.]: Novatec, 2007.

LACERDA, E. G. M. de; CARVALHO, A. C. P. L. F. de. *Introdução aos Algritmos Genéticos*. 2015. <http://www.leca.ufrn.br/~estefane/metaheuristicas/ag.pdf>. Acessado em 1 de Agosto.

LAUDON, K. C.; LAUDON, J. P. *Sistemas de informação gerenciais.* 7º. ed. [S.l.]: Pearson, 2009.

LINDEN, R. *Algoritmos genéticos*. 1º. ed. Rio de Janeiro: Ciência Moderna, 2012.

LUCKNOW, D. H.; MELO, A. A. de. *Programação Java para Web*. 1º. ed. São Paulo: Novatec, 2010.

LUGER, G.; STUBBLEFIELD, W. A. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* 2º. ed. [S.l.]: Palo Alto, 1993.

LUQUE, L.; SILVA, R. Algoritmos genéticos em java, conceitos e aplicação. *Java Magazine*, Rio de Janeiro, v. 82, p. 44–55, 2010.

MANZONI, A. Desenvolvimento de um sistema computacional orientado a objetos para sistemas elÉtricos de potÊncia: AplicaÇÃo a simulaÇÃo rÁpida e anÁlise da estabilidade de tensÃo. In: . [S.l.: s.n.], 2005.

MARCONI, M. A.; LAKATOS, E. M. *técnicas de Pesquisa*. 7º. ed. [S.l.]: Atlas, 2009.

MELANIE, M. *An introduction to genetic algorithms*. 1º. ed. London: Massachusetts Institute of Technology, 1999.

ORACLE. *JavaServer Faces Technology Overview*. 2015. <http://www.oracle.com/technetwork/java/javaee/overview-140548.html>. Acesso em 15 de janeiro de 2015.

ORACLE. *O que é Java?* 2015. <https://www.java.com/pt_BR/download/whatis_java.jsp>. Acessado em 12 de Fevereiro.

PRESSMAN, R. *Engenharia de Software*. McGraw Hill Brasil, 2011. ISBN 9788580550443. Disponível em: <http://books.google.com.br/books?id=y0rH9wuXe68C>.

PRICE, J. *Oracle Database 11g SQL*: Domine sql e pl/sql no banco de dados oracle. 1º. ed. Porto Alegre: bookman, 2008.

ROSS, C. L.; BORSOI, B. T. Uso de primefaces no desenvolvimento de aplicações ricas para web. 2012.

SANTOS, J. C. et al. Seleção de atributos usando algoritmos genéticos para classificação de regiões. In: *XIII Simposio Brasileiro de Sensoriamento Remoto, Florianópolis, Brasil. INPE-Instituto Nacional de Pesquisas Espaciais*. [S.l.: s.n.], 2007. p. 6143–6150.

SANTOS, R. *Introdução à Programação Orientada a Objetos usando Java*. 3º. ed. Rio de Janeiro: Campus, 2003.

SCHILDT, H. *The complete reference Java*. 7º. ed. Nova York: MC Graw Hill Osborne, 2007.

SILVA, E. E. d. *Otimização de estruturas de concreto armado utilizando algoritmos genéticos.* Tese (Doutorado) — Universidade de São Paulo, 2001.

VUKOTIC, A.; GOODWILL, J. *Apache Tomcat 7*. 1º. ed. Nova York: Apress, 2011.

ZANELLA, L. C. H. *Metodologia de Estudo e de Pesquisa em Administração.* 1º. ed. Florianópolis: CAPES, 2009.