CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Departamento de Pesquisa e Pós-Graduação
Curso de Mestrado em Modelagem Matemática e Computacional

UMA SOLUÇÃO DO PROBLEMA DE HORÁRIO ESCOLAR VIA ALGORITMO GENÉTICO PARALELO

Dissertação apresentada ao Curso de Mestrado em Modelagem Matemática e Computacional do CEFET-MG, como parte dos requisitos exigidos para a obtenção do título de Mestre em Modelagem Matemática e Computacional.

por

Eduardo Luiz Miranda Lobo
Bacharel em Ciência da Computação (PUC-MG)

Orientador: Prof. Dr. Sérgio Ricardo de Souza

Belo Horizonte, 28 de outubro de 2005.
Centro Federal de Educação Tecnológica de Minas Gerais

AGRADECIMENTOS

Parabéns ao amigo Sérgio de Oliveira, que deu vida a este trabalho. Obrigado pela oportunidade de dar continuidade ao mesmo e pela sua imprescindível ajuda, apoio e estrutura proporcionada durante o decorrer do mesmo. Sua competência, prestatividade e amizade foram fundamentais no alcance de meus objetivos.

A meu pai Lobinho (in memória), que nunca mediu esforços para ver seus filhos bem formados. Obrigado, pai, por tudo e, principalmente, pela maior riqueza que me deixou: sua índole e moral inalabaláveis!

A minhas filhas, Isabella e Gabriella. Vocês são a razão de meu viver.

A todos os amigos que me apoiaram e torceram por mim. Em especial ao amigo Wendell Taveira.

RESUMO

Este trabalho apresenta o uso de uma solução multiplataforma, usando *Java* e *Corba*, para implementar uma heurística de otimização de quadro de horários. É apresentada uma utilização da metodologia de algoritmos genéticos para desenvolver uma solução de um quadro de horários do grupo professor-turma. O problema consiste na alocação de professores por turma, de forma que mais que um professor não seja associado a uma mesma turma em um mesmo horário e, além disso, todas as restrições envolvidas no problema sejam satisfeitas. O trabalho utiliza princípios de programação paralela, inerentes ao problema de interesse, e será executado em um *cluster* de computadores presentes em uma rede local. O objetivo é a diminuição do tempo computacional utilizado na determinação de uma solução viável. O método desenvolvido é aplicado ao estudo do caso do campus Gigante, da Universidade Presidente Antônio Carlos – UNIPAC, situado em Conselheiro Lafaiete-MG. O problema é modelado e as especificidades dessa instituição de ensino quanto à construção de um quadro de horários são apresentadas e discutidas.

ABSTRACT

This work presents the use of a multi-platform solution using Java and Corba to implement a heuristic of optimization in the construction of a timetable. The method studied here will be the approach of genetic algorithms. Itis aimed to develop a timetabling solution of the class-teacher group, that is, distribution of teachers in classes so that there is no more than a teacher in the same classroom and at the same time, and all the restrictions involved in this problem would be satisfied. The case study will be the Gigante Campus of the Universidade Presidente Antônio Carlos – UNIPAC – in Conselheiro Lafaiete, MG, where there are graduation courses during the day and the night. These courses have a four-hour-duration a day. There are also professors who belong to the staff of more than one course, work in the hourly regime, and also work in another school in the same shift. The work uses parallel programming principles inherents to the problem and it was executed at one computer clusters presents at a local network. This way, it is aimed to improve the computational time of the technique which has been studied in several problems of combinatory optimization and, at the same time , enrichs its practice and enlarge the domain of the problem, as well as the metaheuristic “genetic algorithms”.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

CAPÍTULO I INTRODUÇÃO

A formulação de um quadro de horários que atenda aos interesses de professores e alunos e utilize, da melhor forma possível, os recursos disponíveis, é uma questão primordial na administração escolar, em qualquer nível de ensino. Trata-se, assim, de um problema clássico, que historicamente vem sendo solucionado ou de forma artesanal ou através de soluções empíricas, típicas apenas da própria instituição que as desenvolveram.

A solução artesanal desse tipo de problema é restringida principalmente pelo crescimento das próprias instituições. De fato, o fenômeno da massificação do ensino em seus diversos níveis é relativamente recente em nosso país. Deve-se, contudo, ter claro que, para os diversos níveis de ensino (fundamental, médio ou superior), a vaga da absorção de maior número de alunos se deu em tempos notadamente diferentes, podendo mesmo ser demarcada a partir de gerações de estudantes.

Os anos 90, em particular, trouxeram um forte aumento das matrículas nos níveis fundamental e médio e, principalmente, o grande e rápido crescimento na oferta de ensino superior, tanto no segmento privado quanto no segmento público. Em particular o primeiro, devido às medidas de liberalização na abertura de cursos e de instituições de ensino superior levadas a efeito pelo governo Fernando Henrique Cardoso, sofreu um crescimento explosivo.

Nesse contexto, a correta administração dos interesses de professores e de alunos, expressa em um quadro de horários adequado, passa a ser um problema de grande interesse prático, ao lado de sua importância teórica.

O problema de quadro de horários escolar, em suas várias formulações, despertou o interesse da comunidade acadêmica a partir do início da década de sessenta. Uma das primeiras referências a esse problema é apresentada em [2], em um artigo de Appleby, Black e Newman. Nesse trabalho, os autores enunciavam técnicas para a construção de solução para o problema por meio do emprego de computadores e delineavam a comparação entre o problema de quadro de horários e outros problemas de agendamento conhecidos na época.

Em 1962, Gotlieb [26] apresentou a primeira formulação completa para o problema do quadro de horários para cursos (*Timetabling Problem-Course Scheduling*), ao declarar que o problema consistia em fixar, num determinado período de tempo conhecido, um conjunto de aulas, de forma a atender às exigências acadêmicas estabelecidas por um certo currículo de estudos para um certo grupo de disciplinas. Cada aula visava o atendimento de um único grupo de estudantes e exigia o comparecimento de um único professor à mesma. Nesse trabalho, considerava-se, visando à simplificação do problema, que o número de vezes em que ocorreriam aulas entre estudantes e professor podia ser livremente escolhido e, adicionalmente, que o professor estava totalmente disponível para o atendimento das aulas que lhe fossem agendadas, independentemente do horário das mesmas.

Em 1964, Broder [8] se ateve ao problema do quadro de horários para exames (*timetabling problem-examination scheduling*), tendo, como meta, minimizar os horários conflitantes para a ocorrência de exames finais por parte da comunidade de estudantes envolvida com o currículo de uma instituição superior. A abordagem adotada foi a utilização do algoritmo *Largest Degree First* e, em caso de empate, utilizar o algoritmo de Monte Carlo, no qual a solução com o menor número de conflitos era a escolhida, dentre um conjunto de soluções aleatoriamente obtidas. No mesmo ano, Cole [14] apresentou um algoritmo que viabiliza a introdução de restrições como:

a) certos conjuntos de exames podem ser consecutivos;

b) estabelecimento de ordem de precedência para alguns exames;

c) restrições quanto à capacidade, em número de alunos, das salas de aulas;

d) certos exames podem ser associados a determinados períodos (pela manhã, no caso).

O grande avanço do trabalho de Cole é a utilização de uma matriz de quadro de horários (de exames, no caso) para representar a existência, ou não, de qualquer tipo de conflito entre pares de exames. No trabalho, considera-se que, se dois exames, denominados *i* e *j*, tivessem qualquer tipo de conflito, então, ao elemento *[i,j]* da matriz de quadro de horários deveria ser associado o valor um; caso contrário, deveria

ser associado o valor zero. A abordagem viabilizada pela matriz de quadro de horários permite a visualização dos conflitos de uma maneira simples e de fácil manipulação.

Seguiram-se outras propostas de solução para o problema. Berghuis, Heiden e Bakker [5], em 1964, e Barraclough [4], em 1965, propuseram métodos para a automatização da geração de quadro de horários utilizando computadores como ferramenta para a proposição de soluções práticas. Ainda em 1965, foi publicado o artigo de Csima [18], no qual é apresentada a primeira solução para uma instância especial realística do problema proposto por Gotlieb [26]. Essa solução pode ser implementada por um algoritmo de complexidade polinominal (denotado, matematicamente, como sendo um membro da classe denominada de P, acrônimo de *polynomial*).

Nos anos imediatamente seguintes, a pesquisa em diversas variações e enunciados do problema de quadro de horários escolares se aprofundou, se diversificou e se fundamentou, com especial atenção à modelagem de casos reais. Lions [35] utilizou a similaridade de formulações entre a matriz de incidências de horários no problema de quadro de horários e a matriz de incidências do problema de transporte, para aplicar o método húngaro na geração de quadros de horários escolares. O mesmo autor apresentou, em [36], um programa computacional desenvolvido especialmente para a geração de quadros de horários da *Ontario School*, no Canadá. No mesmo ano (1967), Welsh e Powell [58] estabelecem a relação de similaridade existente entre o problema de quadro de horários e o problema da coloração de grafos. O problema da coloração de grafos foi demonstrado como sendo um exemplar da família dos problemas NP-Completos, por Karp [33], em 1972.

Os resultados teóricos obtidos até o fim da década de 1960 já satisfaziam boa parte das necessidades de aplicações práticas dessa classe de problema existentes naquele momento, desde que os tipos e a quantidade de restrições impostas a uma instância particular do problema não fossem complexas ou numerosas em demasia.

Em 1976, Even, Itai e Shamir [20] demonstraram a implementação proposta por Csima [18] e provaram que o problema geral já estaria enquadrado na classe dos problemas NP-Completos se cada professor pudesse ter, conforme as suas necessidades ou desejos, um determinado número de horários indisponíveis à

realização das aulas a serem estabelecidas. Essa era, indubitavelmente, uma característica existente na maioria das instâncias reais do problema já naquele período.

No início da década de 1980, Schmidt e Stöhlein [49] apresentam um estudo do estado da arte dos trabalhos já realizados sobre o tema até aquele momento. Já Manvel [37] e Metha [38] se ocuparam da coloração de grandes grafos como forma de ataque a complexas e enormes instâncias do problema de quadro de horários. Em [30], Haralick e Elliot propuseram um método para melhorar a eficiência durante o processo de pesquisa na árvore que representa o espaço combinatorial do problema. Em [57], é apresentada uma aplicação do algoritmo de relaxação lagrangeana ao problema de quadro de horários de cursos. Em [59], Werra apresentou uma introdução formal ao problema de quadro de horários. Em [34], Laporte e Desroches focaram seus esforços no problema de quadro de horários para exames.

As abordagens interativas também fizeram parte do elenco de alternativas de solução da década de 1980, como a apresentada por White e Wong [61] em 1988. Na mesma linha, porém mais recente e abrangente, tem-se os trabalhos de Feldman e Golumbic [22], de Gupta e Akhter [29], dentre outros.

A partir do conhecimento de que o problema do quadro de horários pertence à classe dos problemas NP-Completos, foram realizadas diversas tentativas no sentido de dividi-lo em subproblemas e, dessa forma, obter soluções para cada um deles, que pudessem ser eficientemente implementadas, ou seja, obtidas com complexidade polinomial. Assim, esperava-se atingir a extração de uma solução adequada e em tempo satisfatório para grandes instâncias do problema. Em 1995, Cooper e Kingston [17] mostraram que mesmo os subproblemas obtidos a partir da divisão de instâncias reais do problema de quadro de horários pertenciam à classe dos problemas NP-Completos.

Caldeira [10] discute a implementação de dois algoritmos genéticos tratados por, em 1975, por Holland [31], utilizados para resolver o problema do quadro de horários escolar para pequenas escolas, fazendo uma comparação entre os resultados obtidos pelas duas abordagens propostas.

Em sua tese de doutorado, Fang [21] investiga a utilização de algoritmos genéticos para resolver um grupo de problemas de quadro de horários. Nesse trabalho

é proposto um *framework* para a utilização de algoritmos genéticos para a resolução de problemas de quadro de horários no contexto de instituições de ensino. Esse *framework* possui como pontos de flexibilização: a declaração das restrições específicas do problema; a utilização de uma função para avaliação das soluções, também específica para o problema; e a utilização de um algoritmo genético que é independente do problema considerado. Fang mostra que os algoritmos genéticos são bastante efetivos e úteis para a resolução de problemas de quadro de horários e que, quando comparados com os resultados obtidos manualmente, os resultados obtidos por esses algoritmos são mais bem avaliados.

Fernandes [23] classifica as restrições para o problema do quadro de horários escolar em restrições fortes e fracas. Violações às restrições fortes (como, por exemplo, a alocação de um professor em duas salas diferentes em um mesmo horário) resultam em uma solução inválida. Violações às restrições fracas resultam em soluções válidas, porém afetam a qualidade da solução (por exemplo, a preferência dos professores por determinados horários). Fernandes descreve um método para a resolução do problema baseado em algoritmos evolucionários. O algoritmo proposto foi testado utilizando-se uma universidade real com 109 professores, 37 salas de aula, 1131 intervalos de tempo de uma hora cada e 472 aulas. Consegui-se resolver a alocação sem violar as restrições fortes em 30% das execuções. Fernandes compara o algoritmo proposto com outro algoritmo evolucionário que não conseguiu resolver o problema sem violar as restrições fortes em nenhuma das execuções.

Abramson [1] apresenta um algoritmo genético paralelo para o problema. Nesse trabalho é feita uma comparação com os algoritmos genéticos convencionais. Nos experimentos realizados, Abramson considerou instâncias do problema com até trezentas tuplas (professor, disciplina, sala), trinta *slots* de tempo e máquinas com 1, 2, 5, 10 e 15 processadores. Abramson conclui que a abordagem paralela pode ser até 9,3 vezes mais rápida que a abordagem seqüencial, para as instâncias do problema consideradas.

Ribeiro [45] apresenta um algoritmo genético construtivo para o problema do quadro de horários escolar. Nos experimentos realizados, considerou quatro casos de teste reais, de instituições de ensino brasileiras.

O presente trabalho visa apresentar um estudo e a implementação de uma solução do problema de quadro de horários através da metaheurística de algoritmos genéticos. Essa ferramenta vem sendo aplicada para a solução do problema de quadro de horários por diversos autores [1][10][11][15][16][19][21][56]. Contudo, uma desvantagem em sua utilização consiste no seu alto custo computacional. O processo de seleção natural pode demandar muitas gerações de indivíduos, até que aqueles mais bem adaptados comecem a surgir. A partir desta premissa, propõe-se uma abordagem utilizando os recursos de computação paralela, através do qual pode-se ter vários processos populacionais paralelos, que permitam a geração de melhores indivíduos em outras populações, como meio de diversificá-las, fortalecê-las e, assim, acelerar o processo de convergência.

O presente trabalho está organizado, então, da seguinte forma. O capítulo 2 apresenta uma revisão da metodologia de algoritmos genéticos, introduzindo seus conceitos principais e descrevendo os detalhes de sua implementação.

O capítulo 3 trata dos aspectos de computação paralela que são significativos para o melhor entendimento do presente trabalho. Alguns padrões são discutidos e uma descrição mais específica é feita sobre a tecnologia CORBA, sendo abordada sua arquitetura e princípios de funcionamento.

No capítulo 4, uma contextualização dos problemas de quadro de horários é apresentada. Nele, são abordadas as categorias típicas de problemas de quadro de horários. Além disso, cada tipo de problema é formulado e é discutida sua complexidade computacional.

No capítulo 5, é apresentada a arquitetura proposta para a solução do problema objeto de interesse. É o capítulo central da presente dissertação. É mostrada a modelagem computacional do problema, com base no paradigma da programação orientada a objetos. Além disso, a ferramenta computacional construída é completamente detalhada.

O capítulo 6 trata dos testes executados, tendo em vista a análise de caso de interesse, qual seja, o estudo do quadro de horários do campus Gigante da Universidade Antônio Carlos, em Conselheiro Lafaiete – MG. São apresentados os resultados encontrados e análises dos mesmos.

No capítulo 7 são feitas considerações finais e propostas de trabalhos futuros são elencadas.

CAPÍTULO II ASPECTOS GERAIS DE ALGORITMOS GENÉTICOS

Desde a antiguidade, a humanidade vem procurando a imitação de mecanismos existentes na natureza ou para associá-los a tecnologias desenvolvidas pelo próprio homem ou na busca do maior entendimento de seu funcionamento. Em especial no caso de fenômenos biológicos, a diversidade dos empreendimentos humanos visando compreendê-los é tamanha que é parte de nossa própria história de civilização.

Nos séculos XIX e XX, fruto das diversas fases das revoluções tecnológicas, esses esforços, em certo sentido, pela primeira vez ganharam condição científica independente, separando-se das análises religiosas e filosóficas sob a condição humana, para se constituírem em áreas de conhecimento próprias. Atualmente, novas técnicas têm sido inspiradas na natureza ou na biologia de um modo geral, como os algoritmos genéticos ou as redes neurais.

Na segunda metade do século XIX é proposta, por Darwin, a Teoria da Seleção Natural, um dos mais importantes princípios no ramo da evolução, pois defendia a idéia de que na natureza, aqueles seres vivos que melhor se adaptassem tenderiam a sobreviver. A partir desse marco, os fundamentos da teoria evolucionista foram lançados e se, constituem, nos dias de hoje, nos princípios que norteiam as pesquisas sobre o desenvolvimento das espécies ao longo da vida na Terra. Os algoritmos genéticos se inspiram nas análises de comportamento de populações de indivíduos e sua capacidade de adaptação ao meio no qual estão inseridos. A possibilidade de imitação desse comportamento através de algoritmos computacionais deu margem ao surgimento destas novas técnicas de otimização de problemas,

passíveis de serem aplicadas a problemas diversos, sem uma formulação matemática clara.

A origem desses algoritmos pode ser buscada na formulação de Darwin a respeito da evolução das espécies. As teses evolucionistas e a compreensão dos fenômenos da hereditariedade a partir dos estudos de Mendel são os elementos-chave para o desenvolvimento dos algoritmos genéticos. Eles transformam uma população de indivíduos, cada um com um valor associado de adaptabilidade, chamado de aptidão, numa nova geração de indivíduos, usando os princípios de reprodução e sobrevivência dos mais aptos, pela aplicação de operações genéticas como cruzamento e mutação.

Esse capítulo está organizado da seguinte forma: a seção seguinte apresenta um breve histórico do desenvolvimento dos algoritmos genéticos. A seção 2.2 introduz os principais conceitos necessários ao entendimento da metodologia de algoritmos genéticos. A seção 2.3 discute os aspectos principais da construção e implementação dessa classe de algoritmos.

2.1. Breve Histórico do Desenvolvimento de Algoritmos Genéticos

A partir das décadas de 50 e 60, diversos cientistas começaram a estudar sistemas computacionais que buscavam imitar a idéia de evolução de populações. É o início do desenvolvimento dos chamados sistemas computacionais evolucionários, com a premissa que a evolução poderia ser usada como uma ferramenta de otimização para problemas de ordem diversa.

Os principais desenvolvimentos podem ser creditados ao cientista americano John Holland. Ele propôs o desenvolvimento de programas computacionais que incorporassem os princípios da evolução, de modo a possibilitar a solução, via simulação, de problemas complexos, justamente como a natureza o fazia, ou seja, produzindo organismos complexos para resolver o problema de sua sobrevivência. Ele iniciou seu trabalho manipulando cadeias binárias que pudessem representar cromossomos, e cada organismo constituindo uma tentativa da solução do problema.

Seu algoritmo conseguia resolver problemas complexos de uma maneira muito simples. Como a natureza, o algoritmo não sabia o tipo do problema que estava sendo resolvido. Uma simples função de adequação fazia o papel da medida da adaptação dos organismos (cromossomos) ao meio ambiente. Assim, os cromossomos com uma melhor adaptação, medida por essa função, tinham melhor oportunidade de reprodução do que aqueles com má adequação, imitando o processo evolucionário da natureza.

A formulação mais completa de seus primeiros trabalhos foi publicada em 1975, em seu livro "*Adaptation in Natural and Artificial Systems*" [31].

Desde então, a metodologia de algoritmos genéticos tem sido aplicada a uma ampla classe de problemas, nas mais diversas áreas, como otimização de sistemas, metodologias de tomada de decisão, simulação, sistemas de aprendizado, sistemas econômicos.

Como a natureza e os algoritmos genéticos encontram o trajeto mais curto para chegar ao indivíduo mais apto? Numerosos esforços para responder a esta pergunta foram feitos. Entre eles o teorema do *schema*, que é a mais reconhecida [42] [50].

Um *schema* pode ser construído introduzindo-se o símbolo do asterisco (*) no alfabeto dos genes (0 e 1). O asterisco é entendido como "*wildcats*" ou "não se importe" com o símbolo. Se um gene for um "*wildcats*", isso significa que o gene pode ser um 0 ou um 1, e o *schema* representa os cromossomos que combinam estes critérios. Como exemplo, considere o *schema* (*110010100). Há dois cromossomos que combinam este *schema*:

(***1***110010100) ou (***0***110010100)

Um *schema* com dois "*wildcats*" terá quatro opções. Os cromossomos que combinam o *schema* (*11001*100) são:

(***1***11001***1***100), (***0***11001***1***100), (***1***11001***0***100), (***0***11001***0***100)

2.2. Conceitos

2.2.1. Conceitos de Algoritmos Genéticos

Algoritmos genéticos são um conjunto de modelos computacionais inspirados na genética. Estes algoritmos modelam uma solução para um problema específico em uma estrutura de dados como a de um cromossomo e aplicam operadores que recombinam estas estruturas, preservando informações críticas.

Um algoritmo genético é um procedimento que mantém uma população de estruturas (chamadas indivíduos), representando possíveis soluções de um determinado problema. Estas estruturas são, então, avaliadas, para gerar oportunidades reprodutivas, de forma que cromossomos que representam uma solução "melhor" tenham maiores chances de se reproduzirem do que os que representam uma solução "pior". A definição do que seja uma solução “melhor” ou uma solução “pior” é tipicamente relacionada à população atual.

Um algoritmo genético é, assim, qualquer modelo computacional baseado em população que utiliza operadores de cruzamento e mutação para gerar novos pontos amostrais em um espaço de busca. O maior interesse no algoritmo genético está em usá-lo como ferramenta de otimização, pois se trata de uma poderosa ferramenta para busca de soluções de problemas de alta complexidade.

Baseado na analogia com o processo de evolução biológica das espécies, os algoritmos genéticos mantêm uma determinada informação relevante sobre o ambiente e a acumulam durante o período de adaptação. Posteriormente, utilizam tal informação acumulada para minimizar o espaço de busca e gerar novas e melhores soluções dentro de um domínio.

Deve ser observado que cada cromossomo, chamado de indivíduo no algoritmo genético, corresponde a um ponto no espaço de soluções do problema de otimização. O processo de solução adotado nos algoritmos genéticos consiste em gerar, através de regras específicas, um grande número de indivíduos (população).

2.2.2 Principais definições

A seguir, são apresentadas as principais definições relacionadas aos algoritmos genéticos:

- cromossomo ou genótipo: cadeia de caracteres, representando alguma informação relativa às variáveis do problema. Cada cromossomo representa, deste modo, uma solução do problema;

- gen ou gene: é a unidade básica do cromossomo. Cada cromossomo tem certo número de gens, cada um descrevendo certa variável do problema. Podem ser do tipo binário, inteiro ou real;

- população: conjunto de cromossomos ou soluções;

- fenótipo: cromossomo decodificado;

- geração: o número da iteração que o algoritmo genético executa para gerar uma nova população;

- operações genéticas: operações que o algoritmo genético realiza sobre cada um dos cromossomos;

- espaço de busca ou região viável: o conjunto, espaço ou região que compreende as soluções possíveis ou viáveis do problema a ser otimizado. Deve ser caracterizado pelas funções de restrição, que definem as soluções viáveis do problema a ser resolvido;

- função objetivo ou de aptidão: construída a partir dos parâmetros envolvidos no problema. Fornece uma medida da proximidade da solução em relação a um conjunto de parâmetros. A função de aptidão permite o cálculo da aptidão de cada indivíduo e fornecerá o valor a ser usado para o cálculo de sua probabilidade de ser selecionado para reprodução;

- aptidão bruta: saída gerada pela função de aptidão para um indivíduo da população;

- aptidão máxima: melhor indivíduo da população.

Esses são os termos básicos da nomenclatura adotada para o estudo de algoritmos genéticos.

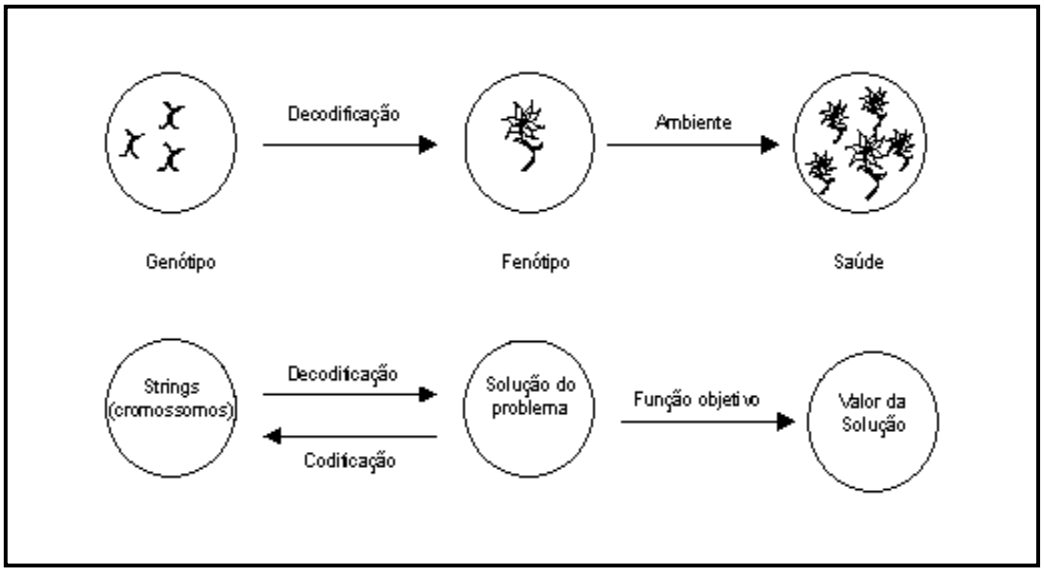


Figura 2.1. Analogia entre genética e algoritmos genéticos.

2.2.3 Analogia entre Genética e Algoritmos Genéticos

Os tópicos relevantes com relação à analogia entre a genética e os algoritmos genéticos (figura 2.1) são listados a seguir:

- Os algoritmos genéticos utilizam o mesmo vocabulário da genética natural.
- Certo número de cromossomos (população) é atualizado a cada iteração (geração) do algoritmo.
- Os indivíduos (genótipos) em uma população são representados por cromossomos.
- Cada cromossomo (fenótipo) é uma solução potencial para o problema.
- No processo de evolução, uma população de cromossomos corresponde a uma pesquisa através do espaço de possíveis soluções.

2.3. Aspectos Principais dos Algoritmos Genéticos

Como primeiro aspecto considerado, tem-se a representação do problema, de maneira que os algoritmos genéticos possam trabalhar adequadamente sobre eles [24]. Pode-se representar as possíveis soluções de um problema no formato de um código genético, que irá definir a estrutura do cromossomo a ser manipulado pelo algoritmo. Essa representação do cromossomo depende do tipo de problema e do que, essencialmente, se deseja manipular geneticamente. Os principais tipos de representação e os problemas aos quais são tipicamente aplicados são mostrados na Tabela 2.1, advinda de [43]:

Tabela 2.1. Tipos de Representação de Cromossomos.

Representação	**Problemas**
Binária	Numéricos, Inteiros
Números Reais	Numéricos
Permutação de Símbolos	Baseados em Ordem
Símbolos repetidos	Grupamento

Tradicionalmente, os indivíduos são representados genotipicamente por vetores binários, nos quais cada elemento de um vetor denota a presença de (1) ou ausência (0) de uma determinada característica, ou seja, o seu genótipo [24] [12].

Definida a representação do problema, a execução do algoritmo pode ser resumida nos seguintes passos:

- Escolhe-se uma população inicial, normalmente formada por indivíduos criados aleatoriamente;
- Avalia-se toda a população de indivíduos segundo algum critério, determinado por uma função, que avalia a qualidade do indivíduo (função de aptidão ou "*fitness*");
- Em seguida, através do operador de seleção, são escolhidos os indivíduos de melhor valor (dado pela função de aptidão) como base para a criação de um novo conjunto de possíveis soluções, chamado de nova geração;

- Esta nova geração é obtida pela aplicação, sobre os indivíduos selecionados, de operações que misturem suas características (genes), através dos operadores de cruzamento (*crossover*) e mutação;

Estes passos são repetidos até que uma solução aceitável seja encontrada ou até que o número predeterminado de passos seja atingido ou, então, até que o algoritmo não consiga mais melhorar a solução já encontrada.

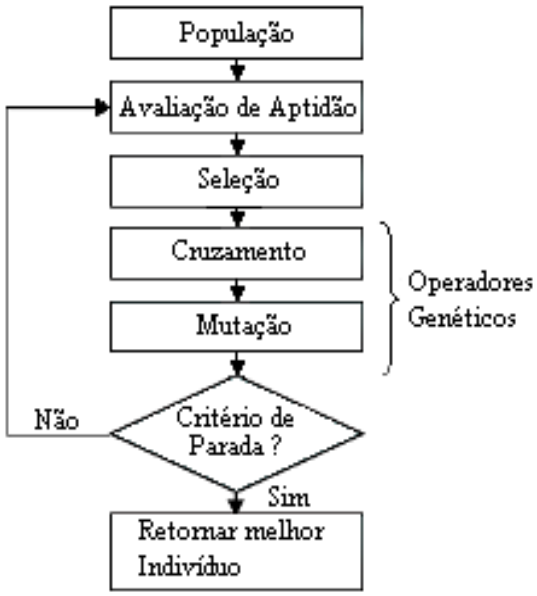


Figura 2.2. Estrutura básica de um Algoritmo Genético.

A estrutura funcional do algoritmo está representada na figura 2.2, sendo descrita, em maiores detalhes, a seguir.

2.3.1. População

A população de um algoritmo genético é o conjunto de indivíduos que estão sendo cogitados como solução e que serão usados para criar o novo conjunto de indivíduos para análise. O tamanho da população pode afetar o desempenho global e a eficiência dos algoritmos genéticos. Por exemplo, populações que são muito pequenas têm grandes chances de perder a diversidade necessária para convergir a uma boa solução, pois fornecem uma pequena cobertura do espaço de busca do problema. Por outro lado, uma grande varredura do espaço de soluções gera uma grande população, que pode prejudicar a comportamento computacional do problema. Segundo [39], "*uma implementação de um algoritmo genético começa com uma população inicial de*

cromossomos formada de forma aleatória. Essas estruturas são avaliadas e associadas a uma probabilidade de reprodução, de tal forma que as maiores probabilidades são associadas aos cromossomos que representam uma melhor solução para o problema de otimização do que àqueles que representam uma solução pior".

2.3.1.1. Indivíduos

Uma das principais formas de representação de problemas é fazer com que cada atributo seja uma seqüência de bits e o indivíduo seja a concatenação das seqüências de bits de todos os seus atributos.

A codificação para representar um indivíduo, usando o próprio alfabeto do atributo que se quer representar (letras, códigos, números reais, etc.), também é muito utilizada. Diversas outras formas são possíveis, mas, normalmente, a forma mais apropriada está fortemente ligada ao tipo de problema.

Como exemplos, na literatura são descritas as seguintes formas [13]:

- Vetores de números inteiros ou de números reais (2,345; 4,3454; 5,1; 3,4);
- Cadeias de bits (111011011).

2.3.2. Avaliação de Aptidão (*Fitness*)

Neste componente é calculado, por intermédio de uma determinada função, o valor de aptidão de cada indivíduo da população. Cada indivíduo é uma entrada para uma ferramenta de análise de desempenho, cuja saída fornece medidas que permitem ao algoritmo genético o cálculo da aptidão desse indivíduo. Ainda nessa fase, os indivíduos são ordenados conforme a sua aptidão [39]. Este é o componente mais importante de qualquer algoritmo genético. É através desta função que se mede quão próximo um indivíduo está da solução desejada ou quão boa é esta solução.

A avaliação é feita através de uma função que melhor representa o problema e tem, por objetivo, fornecer uma medida de aptidão de cada indivíduo na população corrente, que irá dirigir o espaço de busca.. A função de avaliação é, desse modo, específica para cada problema. No exemplo a seguir, a função matemática $f(x) = x^2$

mede a aptidão de cada indivíduo. Na Tabela 2.2, C1 é um indivíduo mais apto que C2 [43].

Tabela 2.2. Exemplo de Função de Aptidão

Indivíduo	**Cromossomo**	***x***	***f(x)***
C1	001001	*9*	*81*
C2	000100	*4*	*16*

A função de aptidão pode ser facilmente encontrada em alguns casos, mas pode ser de difícil definição quando existem fatores de restrições e penalidades internos, ou uma combinação de diferentes objetivos na mesma função.

Tabela 2.3. Exemplo de Função de Aptidão e Função de Aptidão Relativa

Indivíduo	**Aptidão (f_{apt})**	**Aptidão Relativa (f_{rel})**
1	1	0,025
2	3	0,075
3	4	0,100
4	6	0,150
5	7	0,175
6	9	0,225
7	10	0,250

Para alguns métodos de seleção, é desejável que o valor de aptidão de cada indivíduo seja menor que 1 e que a soma de todos os valores de aptidão seja igual a 1 ($f_{apt} < 1$ e $\Sigma(f_{apt}) = 1$). Portanto, para cada indivíduo, é calculada a aptidão relativa (f_{rel}). A aptidão relativa para um dado indivíduo é obtida dividindo-se o valor de sua aptidão pela soma dos valores de aptidão de todos os indivíduos da população, conforme mostrado na Tabela 2.3.

É essencial que a função de aptidão seja muito representativa e diferencie, na proporção correta, as "más" soluções das "boas" soluções. Se houver pouca precisão na avaliação, uma ótima solução pode ser posta de lado durante a execução do algoritmo, além de despender recursos computacionais em um espaço de busca pouco promissor.

2.3.3. Seleção

Dada uma população em que a cada indivíduo foi atribuído um valor de aptidão, o processo de seleção escolhe, então, um subconjunto de indivíduos da

população atual, gerando uma população intermediária. Existem vários métodos para selecionar os indivíduos sobre os quais serão aplicados os operadores genéticos. Dentre eles, serão descritos ométodo de seleção por Roleta; o método de seleção por Torneio; o método da Amostragem Universal Estocástica; e o método da seleção Elitista [24]. Vale notar que, segundo [48], "*o princípio básico do funcionamento dos algoritmos genéticos é que um critério vai fazer com que, depois de muitas gerações, o conjunto inicial de indivíduos gere indivíduos mais aptos. Cada indivíduo tem uma probabilidade de ser selecionado proporcional à sua aptidão. A maioria dos métodos de seleção é projetada para escolher, preferencialmente, indivíduos com maiores notas de aptidão, embora não exclusivamente, a fim de manter a diversidade da população*".

2.3.3.1. Método da Seleção por Roleta

No método de seleção por Roleta, que é muito utilizado, indivíduos de uma geração (ou população) são escolhidos para fazer parte da próxima geração, através de um sorteio de roleta. Cada indivíduo da população é representado, na roleta, proporcionalmente ao seu índice de aptidão. Dessa forma, para indivíduos com alta aptidão, é dada uma porção maior da roleta, enquanto aos indivíduos de aptidão mais baixa é dada uma porção relativamente menor. A roleta é girada um determinado número de vezes, dependente do tamanho da população. A cada giro da roleta, um indivíduo é apontado pela seta e selecionado. Aqueles indivíduos sorteados na roleta são escolhidos como indivíduos que participarão da próxima geração e são inseridos na população intermediária [24][12]. A figura 2.3 ilustra esse método.

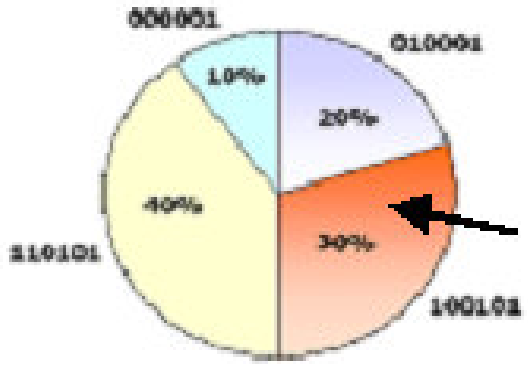


Figura 2.3. Método de Seleção por Roleta

Um exemplo da implementação deste método é mostrado na figura 2.4.

Inicio
T = soma dos valores de aptidão de todos os indivíduos da população;
Repita *N* vezes para selecionar os *n* indivíduos;
r = valor aleatório entre *0* e *T*;
Percorra seqüencialmente os indivíduos da população, acumulando em *S* o valor de aptidão dos indivíduos já percorridos;
Se *S* >= *r* então:
Selecione o indivíduo corrente;
Fim se.
Fim Repita.
Fim.

Figura 2.4. Algoritmo básico do método de seleção por Roleta.

O método da roleta tem a desvantagem de possuir uma alta variância, podendo levar a um grande número de cópias de um bom cromossomo, diminuindo a variabilidade da população. Uma alternativa seria utilizar somente a posição ("*ranking*") de cada indivíduo na população. Mantendo a população ordenada por valores decrescentes da aptidão, a probabilidade de seleção de um indivíduo para a etapa de recombinação cresce com o seu "*ranking*", ou seja, o primeiro do "*ranking*" tem maior probabilidade de seleção [52].

2.3.3.2. Método da Seleção por Torneio

Um outro método é a seleção por Torneio, na qual um número *n* de indivíduos da população é escolhido aleatoriamente para formar uma sub-população temporária. Desse grupo, o indivíduo selecionado dependerá de uma probabilidade *k*, definida previamente.

O cromossomo com maior aptidão dentre estes *n* cromossomos é selecionado para a população intermediária. O processo se repete até que a população intermediária seja preenchida [24].

Um exemplo básico da implementação deste algoritmo é mostrado na figura 2.5, para n = 2:

```
Início
   k = 0.75
   Repita N vezes
            Escolha 2 indivíduos da população aleatoriamente;
            r = valor aleatório entre 0 e 1;
            Se r < k
                     O melhor indivíduo é escolhido;
            Senão
                     O pior indivíduo é escolhido;
            Fim se.
   Fim repita.
Fim.
```

Figura 2.5. Algoritmo básico do método de seleção por Torneio.

Este método é o mais utilizado, pois oferece a vantagem de não exigir que a comparação seja feita entre todos os indivíduos da população. A Tabela 2.4 mostra um exemplo de utilização do método de seleção por torneio[56].

Tabela 2.4. Grau de Aptidão para o Método de Seleção por Torneio

Indivíduo	***f(x)***
1	*169*
2	*576*
3	*64*
4	*361*

Baseado na Tabela 2.4, suponha que sejam sorteados os indivíduos 1 e 2. Nesse caso, quem vence o torneio é o indivíduo 2, pois seu grau de aptidão é o maior. Devido a esse fato, este indivíduo é selecionado para cruzamento. O método possui a grande vantagem da não-geração de super-indivíduos, pois a chance do indivíduo com maior grau de aptidão ser selecionado é a mesma, independentemente de seu grau de aptidão ser alto. No exemplo ilustrado na Tabela 2.4, a chance do indivíduo 2 ser selecionado é *1/4*, pois, se for sorteado, independentemente de quem seja seu rival, ele vencerá o torneio. Se o grau de aptidão fosse 3000 ao invés de 576, as chances de seleção continuariam as mesmas, para esse método. Já no método da roleta, ao contrário, o intervalo de seleção iria aumentar muito e por isso a chance do indivíduo ser selecionado também iria ser bem maior.

2.3.3.3. Método da Amostragem Universal Estocástica

O método da Amostragem Universal Estocástica ou *SUS* (*Stochastic Universal Sampling*) pode ser considerado como uma variação do método da roleta, na qual, ao invés de uma única agulha, são colocadas *n* agulhas igualmente espaçadas, sendo *n* o número de indivíduos a serem selecionados para a próxima geração. Dessa forma, a roleta é girada uma única vez, ao invés de *n* vezes, selecionando assim os indivíduos [24].

Evidentemente, os indivíduos cujas regiões possuem uma maior área terão maior probabilidade de serem selecionados por várias vezes. Conseqüentemente, a seleção de indivíduos pode conter várias cópias de um mesmo indivíduo, enquanto outros podem desaparecer.

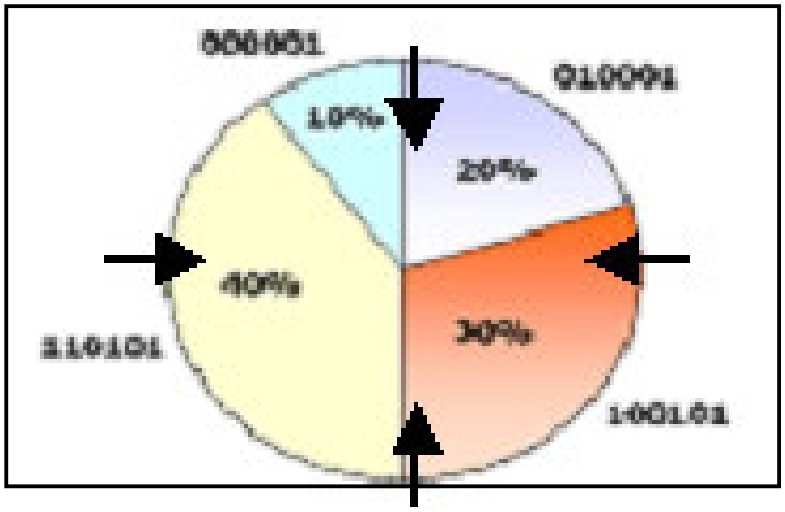


Figura 2.6. Método da Amostragem Universal Estocástica [24].

2.3.3.4. Seleção Elitista

O modelo de seleção elitista normalmente é acoplado a outros métodos de seleção, na tentativa de se aumentar a velocidade de convergência do algoritmo, bem como em aplicações nas quais possa ser necessário o seu emprego isoladamente. Esta técnica consiste em substituir o(s) pior(es) cromossomo(s) da nova geração pelo(s) melhor(es) da antiga [3].

O processo simplesmente copia os *n* (*n*=1) melhores indivíduos da população corrente para a próxima geração, garantindo que estes cromossomos não sejam destruídos nas etapas de recombinação e mutação. Na maioria das implementações,

normalmente, pelo menos o elitismo do melhor indivíduo é utilizado. O método mais utilizado é monitorar apenas um único cromossomo, que melhora muito o desempenho.

A principal vantagem do elitismo é garantir que não se perca o melhor indivíduo durante o processo evolutivo e, com isto, gerar uma seqüência na qual o resultado não piora. Sua desvantagem é a possibilidade de forçar a busca, pela presença de mais uma cópia do melhor indivíduo, na direção de algum ponto ótimo local que tenha sido descoberto antes do global, embora um algoritmo genético possa escapar de tais armadilhas. Uma alternativa é guardar separadamente a melhor solução encontrada durante a evolução, para, no final da execução, designá-la como o indivíduo ótimo encontrado, mesmo que ele não esteja presente na última geração da execução. Segundo [3], "*outras variações da técnica podem ser empregadas, como a que substitui o pior cromossomo da nova população apenas se ele for pior do que o melhor cromossomo da antiga população. Isso garante que a substituição nunca irá diminuir o valor médio da nova população*".

2.3.4. Operadores Genéticos

A reprodução é uma etapa inspirada na natureza e tem, por objetivo, criar novas soluções na população. São utilizados operadores advindos da reprodução humana, tais como os operadores de recombinação e mutação.

O principio básico dos operadores genéticos é transformar a população através de sucessivas gerações, estendendo a busca até chegar a um resultado satisfatório. Os operadores genéticos são necessários para que a população se diversifique e mantenha características de adaptação adquiridas pelas gerações anteriores.

Durante a fase de reprodução de um algoritmo genético, selecionam-se indivíduos da população que serão recombinados para formar descendentes, que, por sua vez, constituirão a geração seguinte. Os pares são selecionados aleatoriamente, usando-se um método que favoreça os indivíduos melhor adaptados. Logo que forem escolhidos os pares, seus cromossomos se mesclam e se combinam, usando os operadores de cruzamento (*crossover)* e mutação. Eles são utilizados para assegurar

que a nova geração seja totalmente nova, porém, mantendo características de adaptação adquiridas pelas populações anteriores.

Para [24], "*o princípio básico dos operadores genéticos é, então, transformar a população por meio de sucessivas gerações, estendendo a busca até chegar a um resultado satisfatório*".

Os operadores de cruzamento e de mutação têm um papel fundamental em um algoritmo genético. Na figura 2.7, é mostrado um exemplo de algoritmo genético. Durante esse processo, os melhores indivíduos podem ser coletados e armazenados para avaliação [12]. Nesse algoritmo, as seguintes variáveis são utilizadas:

- t: tempo atual;
- d: tempo determinado para finalizar o algoritmo;
- P: população.

```
Procedimento AG
{t = 0;
inicia_população (P,t);
avaliação (P,t);
repita até (t = d)
    {t = t + 1;
    seleção_dos_pais (P,t);
    recombinação (P,t);
    mutação (P,t);
    avaliação (P,t);
    sobrevivem (P,t); }
}
```

Figura 2.7. Algoritmo básico do uso dos operadores.

As sub-seções a seguir discorrem a respeito dos operadores cruzamento (*Crossover*) e mutação.

2.3.4.1. Operador Cruzamento (*Crossover*)

Este operador é utilizado após a seleção do indivíduo. Esta fase é marcada pela troca de segmentos entre "casais" de cromossomos, selecionados para dar origem a novos indivíduos, que formarão a população da próxima geração. Esta mistura é feita tentando imitar a reprodução de genes em células. Trechos das características de um indivíduo são trocados pelo trecho equivalente do outro. O resultado desta operação é

um indivíduo que, potencialmente, combine as melhores características dos indivíduos usados como base.

A combinação dos genes responsáveis pelas características do pai e da mãe possibilita o surgimento de infinitas possibilidades de tipos diferentes, fornecendo um vasto campo de ação para a seleção e aumentando a velocidade do processo evolutivo. O *crossover* consiste em dividir aleatoriamente os cromossomos, produzindo segmentos anteriores e posteriores que realizam um intercâmbio para obter novos cromossomos (descendentes).

As três formas mais comuns de reprodução em algoritmos genéticos são o cruzamento em um ponto, o cruzamento em dois pontos e o cruzamento uniforme, que serão detalhados a seguir.

2.3.4.1.1. Cruzamento em um ponto

Com um ponto de cruzamento (*single-point crossover*), seleciona-se aleatoriamente um ponto de corte do cromossomo e, a partir desse ponto, se realiza a troca de material cromossômico entre os dois indivíduos. Sendo aplicado esse cruzamento, os pais trocam suas caldas, gerando dois filhos; caso contrário, os dois filhos serão cópias exatas dos pais.

Cada um dos dois descendentes recebe informação genética de cada um dos pais. Um exemplo nesse sentido pode ser observado na figura 2.8, utilizando cromossomos de 8 bits. A partir de um número aleatório, divide-se o cromossomo. Uma observação importante a respeito do cruzamento é que podem ser gerados filhos completamente diferentes dos pais e, mesmo assim, contendo diversas características em comum. Outra questão é que o cruzamento não modifica um bit na posição em que os pais têm o mesmo valor, considerada uma característica cada vez mais importante com o passar das gerações.

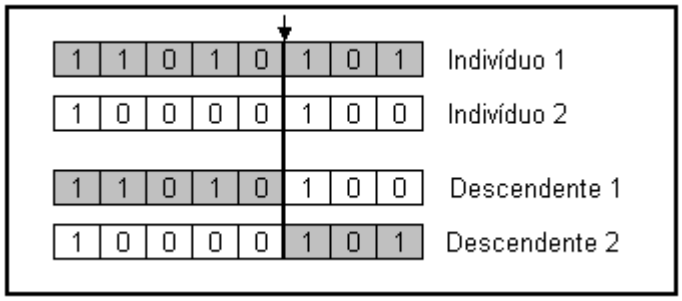


Figura 2.8. Cruzamento em um ponto.

2.3.4.1.2. Cruzamento em dois pontos

Com dois pontos de cruzamento (*two-point crossover*), procede-se de maneira similar ao cruzamento de um ponto. Um dos descendentes fica com a parte central de um dos pais e as partes extremas do outro pai e vice versa, como representado na figura 2.9.

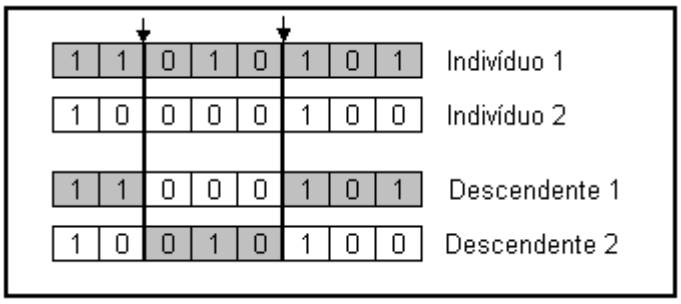


Figura 2.9. Cruzamento em dois pontos.

2.3.4.1.3. Cruzamento uniforme

O cruzamento uniforme (*uniform crossover*) é significativamente diferente dos outros dois cruzamentos apresentados anteriormente. Conforme ilustrado na figura 2.10, primeiramente é criada uma máscara de cruzamento de forma aleatória; posteriormente, cada gene do descendente é criado, copiando-se o gene correspondente de um dos pais, que é escolhido de acordo com a máscara de cruzamento, de modo que, se um certo bit da máscara de cruzamento for 1, o gene correspondente será copiado do primeiro pai; se um certo bit da máscara de cruzamento for 0, será copiado do segundo pai.

O processo é repetido com os pais trocados, para produzir o segundo descendente. Uma nova máscara de cruzamento é criada para cada par de pais [28].

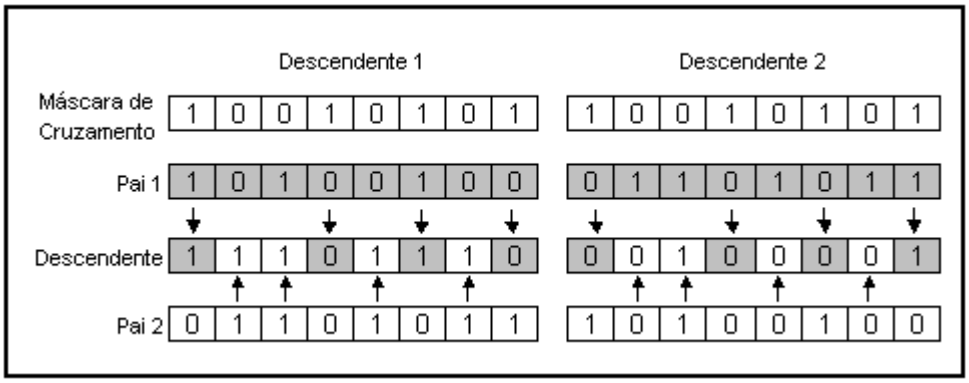


Figura 2.10. Cruzamento uniforme

2.3.4.2. Operador Mutação

A hereditariedade possibilita a estabilidade em sistemas biológicos. Porém, nenhum mecanismo composto de moléculas e sujeito ao impacto do mundo físico pode ser perfeito. Podem ocorrer erros na cópia, que produzem seqüências alteradas de DNA, ou seja, mutações que são perpetuadas.

Mutação pode ser definida como uma mudança na seqüência de pares de base de um gene, mas, às vezes, o termo é usado de maneira mais ampla, de modo a incluir mudanças no número e estrutura dos cromossomos. Pode-se dizer que o cruzamento difere da mutação, porque o cruzamento é, usualmente, uma troca recíproca de estruturas de DNA (genes) que, em si mesma, não são alteradas. Contudo, o cruzamento não é sempre recíproco e pode ocorrer dentro dos limites de um gene e, assim, alterar a seqüência de pares de base. Desse modo, alguns eventos são, na realidade, mutações. A mutação representa a matéria-prima da evolução. Pode-se dizer que, sem esse fator, a vida nunca passaria além de uma bactéria.

A mutação é geralmente vista como um operador de "*background*", responsável pela introdução e manutenção da diversidade genética na população. Esta operação simplesmente modifica aleatoriamente alguma característica de um ou mais genes do cromossomo sobre o qual é aplicada. Esta troca é importante, pois acaba por criar novos valores de características que não existiam, ou apareciam em pequena quantidade na população em análise. Ela trabalha alterando arbitrariamente um ou mais componentes de uma estrutura escolhida entre a descendência, logo após o cruzamento, fornecendo, dessa forma, meios para a introdução de novos elementos na população.

Quando se usa uma representação binária, um bit é substituído por seu complemento (um 0 é substituído por 1 e vice-versa). Este operador é responsável pela introdução de um novo material genético na população de cromossomos, tal como acontece com seus equivalentes biológicos [45].

A figura 2.11 representa a troca de bits, de 1 para 0, como aconteceria no caso de uma representação binária.

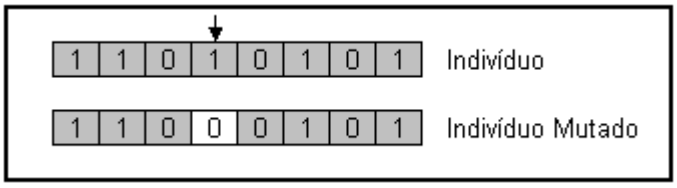


Figura 2.11. Mutação Simples.

O operador de mutação percorre todos os bits do cromossomo e, para cada bit, gera um evento com probabilidade *pm*; se este evento ocorrer, o valor do bit é trocado. Como a probabilidade *pm* é muito baixa ($0 \leq pm \leq 1$), poucos cromossomos são afetados por ela.

Um exemplo de mutação é mostrado na figura 2.12, no qual apenas foi alterado o valor do terceiro bit do segundo cromossomo.

Número	Cromossomo antigo	Número Aleatório	Novo cromossomo
1	0 1 0 1	0,12; 0,70; 0,45;0,92	0 1 0 1
2	1 1 0 1	0,92;0,13;0,06; 0,23	1 1 1 1

Figura 2.12. Exemplo de mutação (*pm* = 0.08)

2.3.5. Critérios de Parada

Diferentes critérios podem ser utilizados para terminar a execução de um algoritmo genético. Como exemplo, podem ser citados os seguintes

- após um dado número de gerações (avaliações), ou seja, um total de ciclos de evolução de um algoritmo genético;
- quando a aptidão média ou do melhor indivíduo não melhorar mais;
- quando as aptidões dos indivíduos de uma população se tornarem parecidas;
- ao conhecer a resposta máxima da função-objetivo;
- no caso de perda de diversidade da população.

CAPÍTULO III UMA ABORDAGEM DE PROGRAMAÇÃO PARALELA

Um sistema computacional distribuído é aquele em que as informações em fase de processamento são distribuídas para vários computadores, em lugar de ficarem confinadas a uma única máquina. Praticamente todos os sistemas com base em grandes computadores são, hoje em dia, sistemas distribuídos [53].

Os componentes de um sistema distribuído podem ser implementados em diversas linguagens de programação e executados em tipos de processadores inteiramente diferentes. Os modelos de dados, a representação das informações e os protocolos de comunicação podem, todos eles, ser diferentes. Portanto, um sistema distribuído exige um *software* que possa gerenciar essas diversas partes e garantir que elas se comuniquem e troquem dados.

Quando se deseja utilizar e implementar aplicações distribuídas, existem vários mecanismos que podem ser utilizados e que, dependendo do tipo e da natureza da aplicação (plataforma operacional e linguagem de programação) devem ser considerados.

Potts e Kopack [44] agruparam estes mecanismos em categorias, entre as quais cabe ressaltar:

- Arquiteturas de RPC (*Remote Procedure Call*) baseadas em *stub*/*skeleton* (CORBA, RMI, DCOM).
- Arquiteturas transacionais tipo HTTP (*servlets*, JSP, ASP, PHP, CGI).

- *Web Services*, que, segundo os autores, se constitui em uma evolução das duas arquiteturas anteriores.

A essa relação cabe, também, incluir os chamados "soquetes", a partir do qual se originaram as RPCs , que evoluíram até mecanismos como a RMI (*Remote Method Invocation*).

3.1. Soquetes

Constituem-se no caminho mais simples de comunicação entre aplicativos localizados em máquinas diferentes. Pela sua simplicidade, porém, não são muito aplicados na manipulação de tipos de dados ou de aplicações complexas, principalmente quando o componente da aplicação está em uma máquina diferente [51].

Soquete é um canal de comunicação através do qual uma aplicação se comunica com outra. Um par de processos (ou *threads*) se comunica em uma rede utilizando um par de soquetes – um para cada processo. Um soquete é formado por um endereço IP concatenado com um número de porta. Em geral, os soquetes utilizam uma arquitetura cliente-servidor. O servidor espera por pedidos de clientes "ouvindo" (*listening*) uma porta específica. Assim que um pedido é recebido, o servidor aceita uma conexão do soquete cliente para completar a conexão.

A comunicação usando soquetes – embora comum e eficiente – é considerada uma forma de comunicação de baixo nível entre processos ou *threads* distribuídos. Um dos motivos é que os soquetes só permitem a troca de um fluxo não estruturado de bytes entre os *threads* em comunicação, sendo responsabilidade da aplicação, do lado cliente ou do lado servidor, impor uma estrutura aos dados.

3.2. Chamadas de Procedimentos Remotos – RPC

Uma Chamada de Procedimento Remoto (RPC – *Remote Procedure Call*) é uma evolução dos soquetes [51]. O mecanismo RPC permite que clientes façam chamadas a procedimentos em serviços remotos como se fossem chamadas de procedimento local. Um cliente pode usar um serviço de diretório para conectar-se a um servidor em tempo de execução. Além disso, o cliente e o servidor podem usar

serviços de segurança para garantir os níveis de autenticação, autorização, integridade e de privacidade. O mecanismo de RPC isola do cliente os detalhes da localização dos servidores na rede; o tipo de plataforma de máquina ou do sistema operacional que executam; as diferenças na representação de dados entre plataformas; e o transporte de rede em uso, permitindo que programas distribuídos trabalhem de forma transparente em sistemas heterogêneos.

3.3. Invocação de Método Remoto – RMI

A invocação de método remoto (RMI - *Remote Method Invocation*) é um recurso Java que permite invocar um método em um objeto remoto. Os objetos são considerados remotos se residirem em máquinas virtuais Java (JVM – *Java Virtual Machine*) distintas. Portanto, o objeto pode estar em uma JVM diferente no mesmo computador ou em um servidor remoto conectado por uma rede.

Na terminologia RMI, o objeto cujo método faz a chamada remota é denominado objeto cliente. O objeto remoto é chamado objeto servidor. Um servidor de uma chamada anterior pode se tornar cliente, quando chamar um método remoto em um objeto residente em outra máquina virtual.

Para tornar os métodos remotos transparentes ao cliente e ao servidor, a RMI implementa o objeto remoto usando *stubs* e *skeletons*. Um *stub* é um *proxy* (representante) do objeto remoto, que reside junto ao cliente. Quando o cliente invoca um método remoto, na realidade ele acessa esse *stub* para o objeto remoto. Esse *stub* no cliente é responsável por criar um pacote, que consiste no nome do método a ser invocado no servidor e nos parâmetros desse método. Esse processo é conhecido como agregação dos parâmetros.

O *stub* envia, então, esse pacote para o servidor, sendo recebido pelo *skeleton* do objeto remoto. O *skeleton* é responsável por efetuar a operação de extração dos parâmetros e por invocar o método desejado no servidor. O *skeleton* agrega o valor de retorno (ou exceção, se houver) em um pacote e retorna-o ao cliente. O *stub* efetua a extração do valor de retorno e o passa para o cliente. O fluxo de informações de uma chamada de método remoto é mostrado na Figura 3.1.

Para acessar um objeto remoto existente no servidor surge um problema: o cliente precisa de um objeto *stub* local para fazer este acesso. Porém, é necessário solicitar um stub. Geralmente, chama-se um método remoto de outro objeto servidor, que é de conhecimento geral e, a partir deste, pode-se obter um objeto *stub* como valor de retorno. Entretanto, o primeiro objeto servidor precisa ser localizado de alguma outra forma.

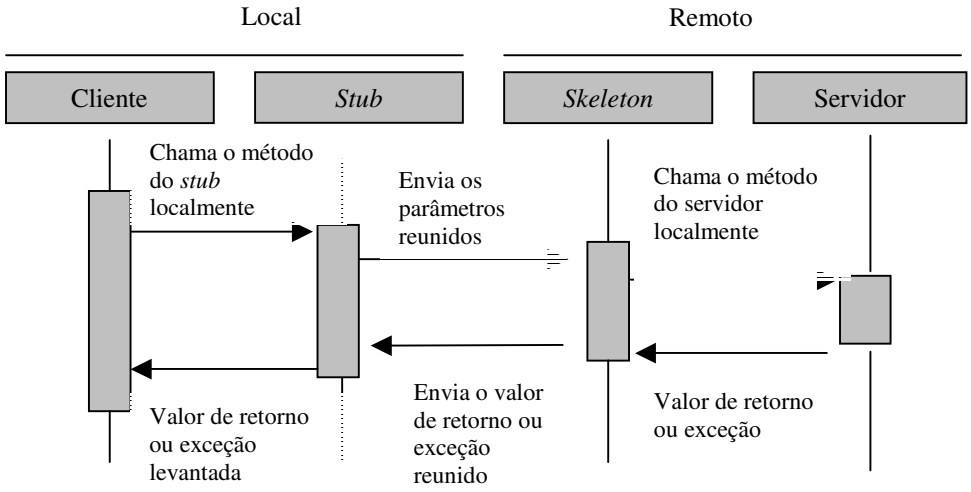


Figura 3.1: Fluxo de informações de uma chamada de método remoto

Para isto, a biblioteca RMI, da empresa *Sun Microsystems Inc.*, fornece um serviço de nomes (RMI *Registry*), que fornece informações sobre a localização de objetos remotos. Através deste serviço, um objeto é associado a um URL de RMI. Essa associação é criada através do método *rebind*( ) e, assim, o objeto registrado na aplicação *rmiregistry* é disponibilizado aos sistemas clientes. A referência feita por qualquer um destes clientes ao URL de um objeto é, na verdade, uma referência ao objeto vinculado.

RMI usa um protocolo proprietário chamado *Java Remote Method Protocol* - JRMP. Este protocolo possui, atualmente, duas versões: a primeira foi implementada no *Java Development Kit* versão 1.1 (JDK 1.1) do RMI , desenvolvido pela empresa *Sun Microsystems Inc.* e requer o uso de classes *skeleton* no servidor. A segunda versão, implementada no Java 2 SDK, foi otimizada e não requer o uso de classes *skeleton*.

3.4. RMI-IIOP

A RMI não é a única escolha para realizar invocações de método remoto em Java. Pode-se usar, também, a *Java Remote Method Invocation over the Internet Inter-ORB Protocol* (RMI-IIOP), que é uma versão especial da RMI, compatível com a especificação CORBA.

A RMI-IIOP não possui alguns dos recursos disponíveis em RMI, como coleta de lixo distribuída, por exemplo. Entretanto, a especificação *Java 2 Platform Enterprise Edition* (J2EE), que inclui a arquitetura *Enterprise Java Beans* (EJB), determina que se use o RMI-IIOP e não a RMI [54] [7].

Na RMI-IIOP são utilizadas classes *skeleton* no servidor.

3.5. A Especificação CORBA

O *Object Management Group* (OMG) é a organização que coordena o desenvolvimento do padrão CORBA. Ela foi fundada em 1989, por um grupo de empresas, com o objetivo de combinar duas tecnologias: *Remote Procedure Calls* (RPC) e orientação a objetos [6].

3.6.1 O Object Request Broker (ORB)

A *Object Management Architecture* (OMA) define as várias facilidades necessárias para a computação distribuída, utilizando-se o conceito da orientação a objetos. O núcleo da OMA é o *Object Request Broker* (ORB), um mecanismo que permite a localização, comunicação e ativação de objetos, sendo o *software* que implementa a especificação CORBA. A especificação CORBA (*Common Object Request Broker Architecture*), desenvolvida com base na OMA, descreve as interfaces e facilidades que devem ser fornecidas pelos ORBs.

Existem várias implementações de ORBs disponíveis no mercado. Entre elas, pode-se citar: ORBIX (da IONA *Technologies*), VisiBroker (da Inprise) e JavaIDL (da JavaSoft).

Para garantir a interoperabilidade entre objetos pertencentes a diferentes implementações de ORBs, tornou-se necessário padronizar o protocolo de comunicação e o formato das mensagens. Assim, foram definidos os seguintes protocolos [46]:

- Protocolo Inter-ORB Geral (GIOP): especifica um conjunto de formatos das mensagens e dados para comunicação entre ORBs;
- Protocolo Inter-ORB Internet (IIOP): especifica como mensagens GIOP são transmitidas numa rede TCP/IP (GIOP+TCP/IP=IIOP). Permite a interoperabilidade entre diferentes implementações de ORBs;
- Protocolo Inter-ORB para Ambiente Específico (ESIOPs): é uma especificação para permitir a interoperabilidade do ORB com outros ambientes (ex: DCOM).

3.6.2. A Linguagem de Definição de Interface (IDL)

Uma das principais vantagens da CORBA é sua independência da linguagem. Os objetos CORBA escritos em uma linguagem de programação podem ativar métodos em um objeto escrito em outra linguagem de programação. Essa independência de linguagem é obtida através do uso da linguagem neutra IDL (*Interface Definition Language*) [46].

Na IDL, não são implementados métodos de objeto – eles são apenas especificados. Mais precisamente, para cada objeto servidor, é codificada uma interface IDL, que lista as constantes, tipos, exceções, atributos e métodos associados ao objeto. Em seguida, o arquivo IDL, que especifica a interface, é compilado e a interface é implementada em uma linguagem de programação específica, como Java ou C++, obtendo, dessa forma, seu objeto servidor [27].

Uma interface descrita em IDL especifica o formato da chamada das operações providas pelo objeto e cada um dos parâmetros necessários para efetuar a operação.

3.6.3. Descrição do Funcionamento do ORB

Usando um ORB, o objeto cliente pode invocar, de forma transparente, um método em um objeto servidor, que pode estar na mesma máquina ou em uma rede. O ORB intercepta a chamada e é responsável pela localização de um objeto que possa implementar a solicitação. É também sua responsabilidade passar os parâmetros a este objeto, invocar seu método e retornar os resultados.

A Figura 3.2 mostra uma requisição sendo transmitida por um cliente para uma implementação do Objeto [40].

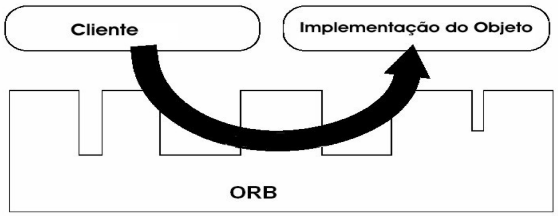


Figura 3.2: Transmissão de uma solicitação através do ORB.

O cliente não tem que estar ciente da localização do objeto, da sua linguagem de programação, do seu sistema operacional e nem de nenhum outro aspecto do sistema que não faça parte da interface do objeto.

O cliente faz a invocação a um objeto através do acesso a uma referência ao objeto, tendo que conhecer, antecipadamente, o tipo do objeto e as características do método a ser executado. Uma referência ao objeto encapsula todas as informações (incluindo a localização) necessárias para se utilizar um objeto CORBA.

As definições de interfaces de objetos podem ser especificadas de duas maneiras: definidas estaticamente através da IDL, ou podem ser adicionadas a um serviço de Repositório de Interfaces, o qual mantém os componentes de uma interface representados como objetos, permitindo, assim, acessos dinâmicos a estes componentes.

Para executar uma solicitação, o cliente pode utilizar a interface de Invocação Dinâmica ou um *stub* IDL, conforme mostrado na Figura 3.3.

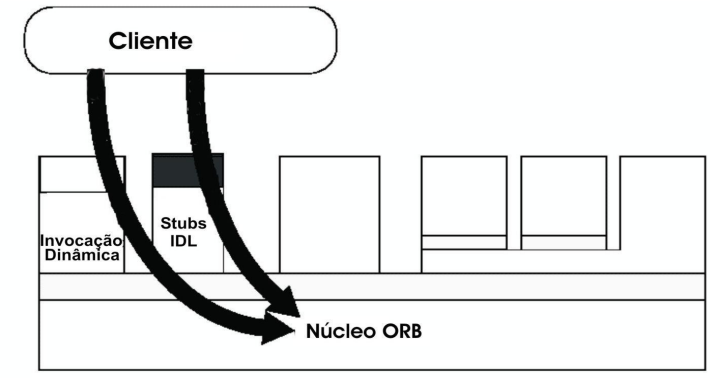


Figura 3.3: Um cliente usando um *stub* ou a Interface de Invocação.

Tanto a interface através de um *stub* quanto a interface dinâmica utilizadas na solicitação devem ter a mesma semântica, de tal maneira que quem recebe esta requisição não é capaz de identificar qual foi a forma invocada.

O ORB localiza a implementação do código do objeto apropriada, transmite os parâmetros e transfere o controle para a Implementação do Objeto através de um *skeleto*n IDL ou de um skeleton dinâmico, como representado na Figura 3.4.

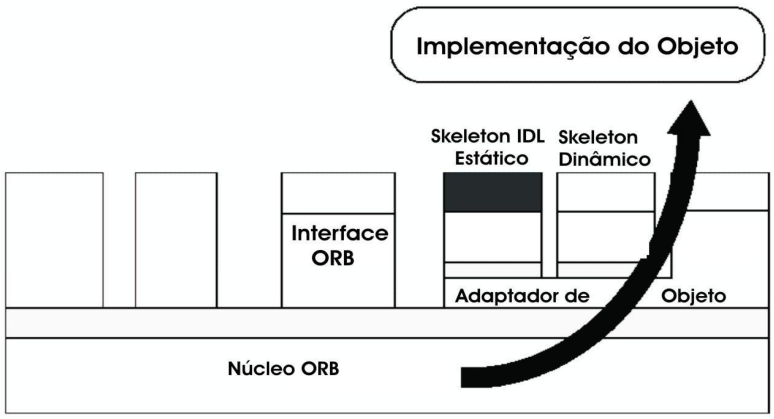


Figura 3.4: Implementação de Objeto recebendo uma requisição.

Ao atender uma requisição, a Implementação do Objeto pode necessitar de determinados serviços oferecidos pelo ORB através do Adaptador de Objeto. Quando a requisição for completada, o controle e os valores dos resultados são retornados ao Cliente, novamente através do ORB.

A Figura 3.5 mostra como as informações de implementação e de interface são geradas e disponibilizadas aos clientes e às Implementações de Objetos. A interface é definida em IDL e/ou obtida do Repositório de Interfaces. A definição IDL

é utilizada para gerar os *stubs* dos Clientes e os *skeletons* das Implementações dos Objetos.

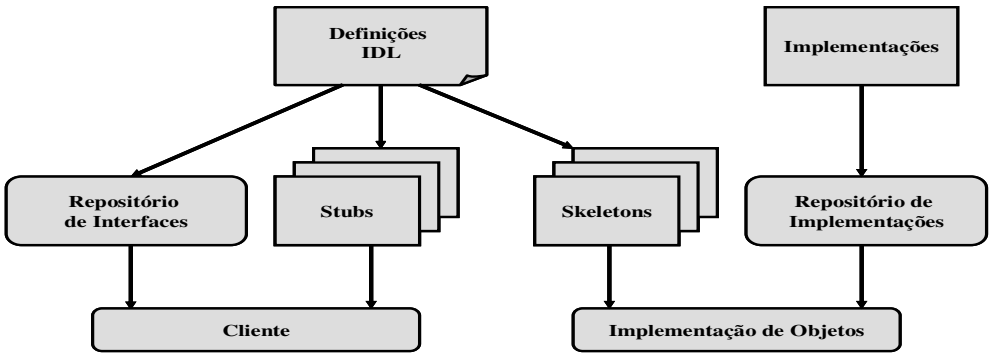


Figura 3.5: Repositório de Interface e Repositório de Implementação.

A informação da Implementação do Objeto é obtida em tempo de instalação e armazenada no Repositório de Implementações, para ser utilizada durante o atendimento das solicitações feitas aos objetos.

Quando o ORB, por algum motivo, não é capaz de completar a invocação, uma resposta do tipo exceção é gerada.

Os clientes acessam os *stubs* como se fossem rotinas de bibliotecas nos seus próprios programas. Desta forma, o programa do cliente vê as rotinas requisitáveis como se fossem rotinas comuns da linguagem de programação.

A Implementação do Objeto interage com o ORB para estabelecer sua identificação, para criar novos objetos e para obter serviços específicos do ORB. Basicamente, este processo é feito através de um Adaptador de Objeto.

Quando um novo objeto é criado, o ORB pode ser notificado, de forma que ele passe a conhecer como encontrar a implementação deste novo objeto. Normalmente, a Implementação de Objeto também se registra como uma nova implementação, além de especificar como dar início ao seu uso, caso ela ainda não esteja em execução.

3.6.4 Os Adaptadores de Objetos

O CORBA especifica um *Basic Object Adapter* (BOA), que deve estar disponível em todo ORB, fornecendo, pelo menos, as seguintes funções [41]:

- Um Repositório de Implementação, que permita a instalação e o registro de aplicativos. Contém também informações que descrevem o aplicativo.
- Mecanismos para a geração e interpretação de referências de objeto; ativação e desativação de implementações de objetos; invocação de métodos e passagem de parâmetros.
- Um mecanismo para a autenticação do cliente que faz a chamada. O BOA não aplica nenhum estilo específico de segurança. Ele garante que, para toda invocação de objeto ou de método, identificará o cliente em nome de quem a solicitação está sendo feita. O que fazer com esta implementação fica a cargo da implementação.
- Ativação e desativação dos objetos de implementação.
- Invocações de métodos através de *stubs*.

A utilização do BOA mostrou algumas limitações e ambigüidades na especificação, conduzindo a uma fraca portabilidade entre aplicações desenvolvidas para diferentes implementações de ORBs. Em função disso, a OMG incluiu, na nova especificação do CORBA, o *Portable Object Adapter* - POA (a partir da versão CORBA 2.2). Trata-se de uma extensão do BOA, que incrementa o suporte à portabilidade entre ORBs implementados por diferentes desenvolvedores.

3.6.5. Os Serviços CORBA

Além de permitir que os objetos se comuniquem pela rede, o OMG publicou um conjunto de serviços de objeto CORBA (conhecido como CORBA *Object Services* – COS) que fornecem capacidades adicionais a seus objetos em rede [6][47]. Esses serviços são oferecidos opcionalmente por fornecedores de produtos CORBA. Entre eles, cabe destacar:

- CORBA *Naming Service* (COS *Naming*): permite pesquisar objetos CORBA por nome.

- *Event Service*: possibilita comunicações assíncronas entre objetos CORBA.

- *Object Transaction Service* (OTS): permite que objetos CORBA realizem transações.

- *Concurrency Control Service*: possibilita que múltiplos clientes interajam simultaneamente com um recurso.

- *Security Service*: adiciona funcionalidades de segurança ao sistema CORBA.

- *Externalization Service*: permite que tipos de dados CORBA sejam armazenados em uma .sistema externo.

- *Interoperable Naming Service*: é um diretório de serviço básico, que permite armazenar, em um repositório central, os nomes relacionados a referências de objetos.

- *Trading Object*: fornece um diretório de serviços flexível, que facilita a localização de objetos através de consultas às suas propriedades.

CAPÍTULO IV O PROBLEMA DO QUADRO DE HORÁRIOS

4.1. Entendendo o Problema do Quadro de Horários

Um problema de quadro de horários (*timetabling problem*) consiste em fixar encontros entre professores e estudantes, dentro de horários pré-estabelecidos, e que devem obedecer a determinadas restrições, que mapeiam políticas diversas, de acordo com cada instituição.

Muitos problemas são solucionados manualmente e isto, de um modo geral, demanda vários dias de trabalho, dada a enormidade de restrições que um problema real de quadro de horários absorve. No entanto, essa massa de trabalho não garante uma solução satisfatória. Pode-se ter o caso de um professor que leciona em mais de uma escola e, em uma solução, ter sido alocado em um dia e horário que colide com a outra instituição em que ele trabalha.

Como o quadro de horários é um problema de amplo espectro de aplicação e sua solução é de difícil implementação, uma atenção considerável tem sido dispendida pela comunidade acadêmica desde a década de 60.

4.2. Factibilidade, otimalidade e complexidade

Uma solução de um problema de busca é aquela que satisfaz a todas as restrições existentes, ou seja, qualquer solução factível é aceita como uma solução para o problema. Já a solução de um problema de otimização é aquela que, dentre todas as soluções factíveis, considera-se como a de menor valor da função objetivo (no caso de problema de minimização) ou de maior valor da função objetivo (no caso de

problema de maximização), em ambos os casos, para todo o espaço de soluções factíveis.

Um problema de quadro de horários pode ser tratado como um problema de busca. No entanto, é, também, um problema de otimização, no qual se procura atender a todas as restrições necessárias (*hard constraints*) e maximizar o atendimento das restrições desejáveis (*soft constraints*).

Neste capítulo será mostrado que o quadro de horários é um problema NP-Completo em quase todas as suas variantes. Assim, soluções exatas são alcançadas somente quando a instância do problema não possuir um grande número de variáveis. Para instâncias com grande número de variáveis, métodos heurísticos são mais aplicáveis.

4.3. Métodos de Solução do Problema

Muitas técnicas de solução do problema de quadro de horários baseiam-se em soluções manuais. Soluções desse tipo podem ser tratadas como heurísticas construtivas, que consistem, basicamente, em alocar aula a aula, até que todo o quadro de horários esteja construído. A idéia principal desta técnica é tratar os casos mais restritivos em primeiro lugar e, após, agendar os horários de aulas com menor número de restrições. Claramente, a definição de "mais restritivo" irá depender da política de restrições da instituição/problema.

Apesar da aparente facilidade das técnicas como a descrita, o crescimento da complexidade dos quadros de horários nas diversas instituições de ensino mostrou a necessidade do desenvolvimento de técnicas que automatizassem a geração de quadro de horários. Assim, diversas técnicas de solução têm sido propostas para esse problema, como, por exemplo, sua formulação como um problema de programação inteira, como problema de fluxo de redes ou usando técnicas de coloração de grafos.

Recentemente, métodos pertinentes ao campo de IA (inteligência artificial) começaram a ser aplicados no tratamento de quadro de horários, dentre os quais podemos citar *simulated annealing*, busca tabu, algoritmos genéticos e satisfação de restrições.

4.4. Tipos de Sistemas Utilizados para a Solução de Quadro de Horários.

Alguns pesquisadores defendem a idéia de que um problema de quadro de horários não pode ser totalmente automatizado, pois:

- as razões que fazem um quadro de horários ser melhor que outro não podem ser facilmente expressadas em um algoritmo;
- a intervenção humana pode influenciar na busca de uma direção promissora que o sistema, por si só, teria dificuldades em encontrar.

Devido a isso, muitos sistemas de geração de quadro de horários possibilitam uma interação com o usuário em maior ou em menor grau. Alguns destes sistemas são denominados sistemas de quadro de horários interativos ou semi-automáticos, devido a larga intervenção do usuário [61].

4.5. Quadro de Horários Escolar

4.5.1. Formulação do Problema

Considere t turmas, na forma t_1, t_2, t_3, ...,t_t, ; p professores, na forma p_1, p_2, p_3, ...,p_p ; e h períodos h_1, h_2, ..., h_h. Considere, ainda, uma matriz de inteiros não-negativos R_{txp}, em que r_{ij} é a carga horária do professor j com a turma i.

O problema consiste em associar aulas a períodos, de forma que não haja sobreposição de professores e/ou turmas, ou seja, que duas ou mais turmas não tenham aulas com o mesmo professor no mesmo período ou que dois ou mais professores não lecionem para a mesma turma no mesmo período.

Uma formulação matemática para esse problema é dada por Werra [60]:

$$\textit{Encontrar} \quad x_{ijk} \qquad (i = 1...t;\ j = 1...p;\ k = 1...h) \quad (4.1)$$

$$\textit{suj.a} \quad \sum_{k=1}^{h} x_{ijk} = r_{ij} \qquad (i = 1...t;\ j = 1...p) \quad (4.2)$$

$$\sum_{j=1}^{p} x_{ijk} \leq 1 \qquad (i = 1...t;\ k = 1...h) \quad (4.3)$$

$$\sum_{i=1}^{t} x_{ijk} \leq 1 \qquad (j = 1...p;\ k = 1...h) \quad (4.4)$$

$$x_{ijk} = 0 \quad ou \quad x_{ijk} = 1 \quad (i = 1...t;\ j = 1...p;\ k = 1...h) \quad (4.5)$$

Nesta expressão, a variável $x_{ijk} = 1$ se o professor j tem aula com a turma i no período k e $x_{ijk} = 0$, em caso contrário. As seguintes observações são importantes:

- A restrição (1) certifica que o professor *j* lecione a carga horária correta para a turma *j*;
- A restrição (2) certifica que não ocorra sobreposição de turma;
- A restrição (3) certifica que não ocorra sobreposição de professor.

Even, Itai e Shamir [20] mostraram que há sempre uma solução para este problema, a menos que um professor ou turma esteja envolvido em mais que *h* aulas. Isto pode ser modelado como:

$$\sum_{i=1}^{t} r_{ij} \leq h \qquad (j = 1...p) \quad (4.6)$$

$$\sum_{j=1}^{p} r_{ij} \leq h \qquad (i = 1...t) \quad (4.7)$$

Even, Itai e Shamir [20] associaram a instância do problema a um multigrafo bipartido em que a solução empregada consistia em encontrar uma seqüência máxima de um conjunto de arestas que não contivessem nós comuns. Neste caso, turmas e professores eram associados a vértices e cada turma t_i era ligada a um professor p_j por arestas r_{ij}. Karp [33] provou que o método roda em tempo polinomial.

Werra [60] mostrou que o problema pode ser reduzido ao problema de coloração de grafos.

4.5.2. Quadro de Horários Escolar com Restrições

Na subseção acima não foi levado em conta a existência de qualquer restrição com relação a indisponibilidades de professores e/ou turmas, assim como não foi considerada alguma pré-associação, como, por exemplo, um professor p_j lecionar para uma turma t_i em um dado horário k.

Junginger [32] propôs a seguinte formulação matemática para o problema com associações segundo o exposto:

$$\textit{Encontrar} \quad x_{ijk} \qquad (i = 1 \ldots t;\ j = 1 \ldots p;\ k = 1 \ldots h) \qquad (4.8)$$

$$\textit{suj a} \quad \sum_{k=1}^{h} x_{ijk} = r_{ij} \qquad (i = 1 \ldots t;\ j = 1 \ldots p) \qquad (4.9)$$

$$\sum_{j=1}^{p} x_{ijk} \leq t_{ik} \qquad (i = 1 \ldots t;\ k = 1 \ldots h) \qquad (4.10)$$

$$\sum_{i=1}^{t} x_{ijk} \leq p_{jk} \qquad (j = 1 \ldots p;\ k = 1 \ldots h) \qquad (4.11)$$

$$x_{ijk} \geq A_{ijk} \qquad (i = 1 \ldots t;\ j = 1 \ldots p;\ k = 1 \ldots h) \qquad (4.12)$$

$$x_{ijk} = 0 \quad ou \quad x_{ijk} = 1 \qquad (i = 1 \ldots t;\ j = 1 \ldots p;\ k = 1 \ldots h) \qquad (4.13)$$

Nesse modelo, as seguintes observações são importantes:

- $x_{ijk} = 1$, se o professor j tem aula com a turma i no período k; e $x_{ijk} = 0$, em caso contrário;
- $t_{ik} = 1$, se a turma i está disponível no horário k; e $t_{ik} = 0$, em caso contrário;
- $p_{jk} = 1$, se o professor j está disponível no horário k; e $p_{jk} = 0$, em caso contrário;
- $A_{ijk} = 1$, se o professor j deve lecionar para a turma i no horário k; e $A_{ijk} = 0$, caso não exista esta pré-associação .

Even, Itai e Shamir [20] mostraram que o problema acima é NP-Completo, através da redução ao problema 3-SAT [25]. Também mostraram que, se todas as turmas estão sempre disponíveis e cada professor está disponível por dois períodos exatos, então o problema torna-se polinomial.

4.5.3. Otimização do Problema

Como citado acima, um problema de quadro de horários pode ser visto como um problema de busca. Em casos reais, no entanto, há a necessidade de se obter a melhor solução possível. Isto o torna um problema de otimização. Desse modo, o problema passa a ser formulado segundo este paradigma. Junginger [32] propôs acrescentar, ao modelo com restrições, a função objetivo:

$$min \quad \sum_{i=1}^{t} \sum_{j=1}^{p} \sum_{k=1}^{h} d_{ijk} \, x_{ijk} \tag{4.14}$$

Nesse caso, tem-se que d_{ijk} é um valor associado ao fato do professor j estar lecionando na turma i no horário k. Este valor será tanto maior, quanto menor for o desejo de que isto aconteça. Portanto, atua como um mecanismo de penalização.

Colorni, Dorigo e Maniezzo [16] formularam uma complexa função objetivo, que mapeia aspectos mais abrangentes de um quadro de horários, baseados nas seguintes quantificações :

- custo didático: consiste em distribuir as aulas por toda a semana;
- custo organizacional: disponibilizar professores para possíveis substituições;
- custo pessoal : estabelecer um dia de descanso para cada professor.

Yoshikawa at al. [62] introduziram uma linguagem de restrições e associaram uma penalidade a cada restrição violada. Sua função objetivo era minimizar todas as penalidades.

4.5.4.Variantes do Problema

Como mencionado anteriormente, um quadro de horários escolar possui muitas variantes, que se adequam, particularmente, a cada instituição. Dentre estas, pode-se citar:

- salas de aulas específicas: há situações em que uma sala de aula específica é necessária para se ministrar um determinado conteúdo ou mesmo pela própria concepção da disciplina. Recursos especiais, como laboratórios ou salas de

música são, por vezes, necessários. Assim, uma restrição adicional é considerada para este caso, já que uma mesma sala não pode ser agendada para duas ou mais turma no mesmo horário *h*;

- vários professores para a mesma disciplina: algumas disciplinas podem ser ministradas por vários professores, de modo compartilhado, ou seja, cada um se responsabiliza por um conjunto de tópicos inerentes a mesma;

- mesma disciplina lecionada para mais de uma turma no mesmo horário *h*: situação comum, como no caso de aulas de educação física, em que podem existir diversas turmas no mesmo horário de aula.

4.6. Quadro de Horários de Curso

O problema do quadro de horários de cursos consiste em agendar um conjunto de aulas para cada curso em um período de tempo pré-determinado e para um dado número de salas, levando em consideração o tamanho das salas para a alocação das turmas. A principal diferença entre o quadro de horários escolar e o quadro de horários de curso é que, no primeiro, as turmas são conjuntos disjuntos de estudantes, ao passo que, no segundo, os cursos podem ter estudantes em comum. Desta forma, para se evitar um conflito de horários, não se deve permitir que dois ou mais cursos com estudantes em comum tenham agendamento de aulas no mesmo horário. A disponibilidade de salas também exerce um importante papel na construção do quadro de horário desta categoria.

O exemplo típico de quadro de horários de curso é a montagem de quadro de horário de sistemas acadêmicos de ensino superior baseados em regime de créditos e matrícula por disciplina. Por outro lado, o quadro de horários escolar tem seu exemplo mais freqüente na construção de quadro de horários de escolas de nível fundamental e médio, seja da rede pública, seja da rede particular. A mesma estrutura pode também ser aplicada para sistemas de ensino superior com regime seriado e matrícula por turma.

4.6.1. Formulação do Problema

O problema do quadro de horários de curso é representado matematicamente considerando-se que existem c cursos, dados por c_1, c_2, ..., c_c. Cada curso c_i consiste de k aulas. Há um conjunto de r currículos, dados por S_1, S_2,...,S_r, formados por cursos com estudantes em comum [60].

Infere-se, então, que os cursos pertencentes ao currículo Si não podem ter suas aulas agendadas para o mesmo período. Tem-se h períodos e a_k é o número máximo de aulas que podem ser agendadas para o período h_i, ou seja, o número de salas disponíveis para aquele período.Segue-se, então, a seguinte formulação :

$$\textit{Encontrar} \quad y_{ik} \qquad (i = 1...c; k = 1...p) \tag{4.15}$$

$$\textit{suj.a} \quad \sum_{k=1}^{p} y_{ik} = k_i \qquad (i = 1...c) \tag{4.16}$$

$$\sum_{i=1}^{c} y_{ik} \leq a_k \qquad (k = 1...p) \tag{4.17}$$

$$\sum_{i \in S_l} y_{ik} \leq 1 \qquad (l = 1...r; k = 1...p) \tag{4.18}$$

$$y_{ik} = 0 \quad ou \quad y_{ik} = 1 \quad (i = 1...c; k = 1...p) \tag{4.19}$$

Nesta expressão, a variável $y_{ik} = 1$ se o curso i possui aula agendada no período k; e $y_{ik} = 0$, em caso contrário. A restrição 4.16 garante que o curso i tenha agendado o número correto de aulas que dizem respeito ao mesmo. Já a restrição 4.17 garante que não seja agendada um número maior de aulas do que o número de salas disponíveis para cada período k. A restrição 4.18 garante a alocação de uma aula de cada curso do currículo S_i por período k.

O problema formulado é NP-Completo, o que pode ser provado através de uma redução ao problema de coloração de grafos. Para tal, veja Werra [60].

4.6.2. Otimização do Problema

Werra [60] acrescentou à formulação a seguinte função objetivo :

$$\max \sum_{i=1}^{c} \sum_{k=1}^{h} d_{ik} y_{ik} \tag{4.20}$$

Nesse caso, d_{ik} é a conveniência de se ter uma aula do curso i no período k. Como o problema já é um NP-Completo, pré-associações e indisponibilidades não o tornam mais complexo.

4.7. Problema de Quadro de Horários de Exames

O problema de quadro de horários de exames consiste em agendar um quadro de exames por curso dentro de um período de tempo h. Assim, é semelhante ao problema de quadro de horários de cursos. Existem situações, inclusive, em que ambos se fundem. Algumas características específicas de um problema de quadro de horários de exames são:

- há, de modo geral, um exame por curso;
- em uma sala, pode ocorrer mais de um exame;
- o número de períodos h pode variar, ao contrário de um problema de quadro de horários de cursos;
- trabalha-se restrições, como :a ocorrência de, no máximo, um exame por dia para cada estudante; e/ou não agendar muitos exames consecutivos para o mesmo estudante;
- a um estudante é possível perder uma aula, devido à sobreposição de horários, mas não se pode afirmar o mesmo no que diz respeito a exames.

4.7.1. Formulação Básica

Segundo Werra [60], a formulação deste problema é similar ao problema de quadro de horários de cursos. Desta forma, considere c cursos, na forma c_1, c_2, ..., c_c e um exame para cada curso c_i. Há um conjunto de r grupos de exames S_1, S_2,...S_r, formados por cursos com estudantes em comum. Infere-se, assim, que os cursos pertencentes a S_l não podem ter seus exames agendados para o mesmo período. Tem-se h períodos e e_k é o número máximo de exames que podem ser agendadas para o período k. O problema é formulado segundo a expressão a seguir:

$$\textit{Encontrar} \qquad y_{ik} \qquad (i = 1...c; k = 1...p) \tag{4.21}$$

$$\textit{suj.a} \qquad \sum_{k=1}^{p} y_{ik} = k_i \qquad (i = 1...c) \tag{4.22}$$

$$\sum_{i=1}^{c} y_{ik} \leq e_k \qquad (k = 1...p) \tag{4.23}$$

$$\sum_{i \in S_l} y_{ik} \leq 1 \qquad (l = 1...r; k = 1...p) \tag{4.24}$$

$$y_{ik} = 0 \quad \textit{ou} \quad y_{ik} = 1 \quad (i = 1...c; k = 1...p) \tag{4.25}$$

Nessa expressão, se $y_{ik} = 1$, então o curso i possui exame agendado no período k; se $y_{ik} = 0$, caso contrário. A restrição (4.21) garante que o curso i tenha agendado um único exame durante todo o período h. A restrição (4.22) limita o número máximo de exames em e_k para o período k. A restrição (4.23) limita a realização de um exame de cada curso do grupo S_l por período k.

4.7.2. Otimização do Problema

Werra [60] considera que o tipo mais comum de restrição desejável para o problema de quadro de horários de exames é evitar que um mesmo estudante faça dois exames consecutivos. Assim, a seguinte função objetivo pode ser esboçada :

$$\min \qquad \sum_{k=1}^{p-1} \sum_{l=1}^{r} \sum_{j \in S_l} y_{ik}\, y_{j(k+1)} \tag{4.20}$$

A função acima penaliza cursos do mesmo grupo S_l sempre que tiverem agendados seus exames consecutivamente. Segundo Metha [38], alguns autores também consideram o número de estudantes envolvidos em cada conflito.

Da mesma forma que o problema de quadro de horários de cursos, pode ser mostrado, por redução ao problema de coloração de grafos, que o problema de quadro de horários de exames é um problema NP-Completo.

CAPÍTULO V ALGORITMOS GENÉTICOS PARALELOS APLICADOS AO PROBLEMA DE QUADRO DE HORÁRIOS

A etapa inicial do problema consiste na especificação. As propostas de algoritmos encontrados levam em consideração características específicas de determinadas escolas, como o número de horários disponíveis para aulas; a rotatividade das turmas pelas salas de aula, além da possibilidade de incluir requisitos necessários (*hard constraints*) e requisitos desejáveis (*soft constraints*) [9]. Requisitos desejáveis podem atender às características pedagógicas (por exemplo, duas aulas seguidas de matemática devem ser evitadas) e preferências de professores (alguns preferem concentrar suas aulas em poucos dias da semana). Requisitos necessários incluem alguns princípios básicos: um professor não pode estar em duas turmas no mesmo período e a turma não pode ter duas aulas ao mesmo tempo. Os requisitos necessários devem ser atendidos para que a solução seja viável. O atendimento dos requisitos desejáveis otimiza o resultado final do sistema.

O uso de algoritmos genéticos busca reduzir o tempo de execução computacional para a resolução do problema de quadro de horários, evitando que se pesquise todo o espaço de possibilidades [19].

Para implementar estes algoritmos, usamos os conceitos de cromossomo, ser e população, implementados em uma linguagem orientada a objetos, multiplataforma e distribuída, sobre um *cluster* de computadores utilizando *Corba*.

5.1. Especificação do Problema

5.1.1. Horários

A solução a ser proposta deve atender a uma escola típica de apenas um turno, seja ele matutino, vespertino ou noturno.

Uma semana deve corresponder a 20 créditos, sendo que cada crédito representa uma aula de 50 minutos. As aulas são agrupadas em ordem 2 (duplas), implicando em 10 possíveis horários para um professor/disciplina.

Tabela 5.1 - Distribuição dos Horários Disponíveis

	Segunda	Terça	Quarta	Quinta	Sexta
Horário 1	i	iii	v	vii	ix
Horário 2	ii	iv	vi	viii	x

A tabela 5.1 mostra a disposição dos horários. Normalmente, entre os dois horários do dia existe um pequeno intervalo.

5.1.2. Professores

Os diversos professores devem ser distribuídos entre as turmas, de forma que um professor não pode lecionar em duas turmas ao mesmo tempo. Este é um requisito necessário. É possível indicar a indisponibilidade de um professor para determinado período, bem como especificar exatamente quais são os horários específicos para um determinado professor lecionar em uma turma. Estes requisitos são desejáveis. Também deve ser possível ao professor optar por concentrar suas aulas durante a semana ou distribuí-las, de forma que não lecione mais de uma aula por dia.

À medida que os requisitos desejáveis vão sendo alcançados, há uma melhora na avaliação daquele horário, ou seja, a solução tende a convergir para a solução ótima.

5.1.3. Turmas

As diversas turmas devem ter preenchidos, no máximo, seus 20 créditos com disciplinas, sempre com aulas geminadas (um período). As disciplinas presentes na grade curricular devem ter um professor responsável. Não existem limites para o número de turmas presentes no quadro.

Cada turma deve ter sua sala de aula associada, de forma que a geração do quadro de horários não seja também responsável pela alocação de salas.

5.1.4. Disciplinas

Uma disciplina é ministrada por um único professor para uma determinada turma. Cada disciplina possui um número de créditos, que corresponde ao total de períodos de aula ministrados para aquela turma durante a semana. Desta forma, disciplinas com, por exemplo, quatro aulas/semana, deverão alocar dois períodos e serão atribuídos a elas dois créditos. Portanto, os créditos atribuídos a uma disciplina serão a metade de sua carga horária semanal.

5.1.5. Requisitos

Dois tipos de requisitos são possíveis de serem especificados: requisitos necessários e requisitos desejáveis.

Requisitos necessários são as restrições ao sistema que devem ser atendidas para que uma solução seja considerada viável. Estas restrições devem ser atendidas na inicialização das estruturas de dados e toda operação realizada sobre o quadro de horários deve respeitar estes requisitos, de forma que o sistema não processe soluções inválidas. Estes requisitos são basicamente: a não ocorrência de sobreposição de professores e sobreposição de turmas, ou seja, não há a possibilidade de dois ou mais professores lecionarem para a mesma turma no mesmo horário; e duas ou mais turmas não podem ter aula com o mesmo professor no mesmo horário.

Requisitos desejáveis são as situações que, se verificadas nas soluções, aumentam sua avaliação, de forma a torná-la uma solução melhor. São considerados bons quadros de horários aqueles que atendam às seguintes restrições:

a) concentração ou distribuição das aulas de um determinado professor de acordo com sua escolha;

b) um Mesmo professor não deve ministrar dois períodos de aulas para a mesma turma, seguidamente;

c) garante o atendimento às restrições de disponibilidade de cada um dos professores.

5.2. IMPLEMENTAÇÃO DA SOLUÇÃO PROPOSTA

5.2.1. Cromossomo

Em algoritmos genéticos clássicos, um cromossomo é representado por uma seqüência de bits. A proposta aqui apresentada é a utilização de uma matriz *M*, de *3* dimensões, associada às turmas, aos professores e aos períodos, de forma que o elemento *M(*[Turma] x [Professor] x [Período]*)* seja um valor binário na forma {0, 1}, de modo que *M[t][pr][pe] = 1* se, e somente se, a turma *[t]* tem aula com o professor *[pr]* no período *[pe]*; caso contrário, assume valor *M[t][pr][pe] =0*.

5.2.2. Inicialização

A população inicial é gerada através de uma heurística construtiva. Cada cromossomo inicial tem seus horários distribuídos de forma que apenas os requisitos necessários sejam atendidos. Como não há a preocupação em atender os requisitos desejáveis, a avaliação de cada indivíduo da população inicial tende a ser baixa.

A tabela 5.2 mostra um exemplo de quadro de horário inicial.

Tabela 5.2 - Quadro de Horários Inicial.

	Segunda	Terça	Quarta	Quinta	Sexta
Horário 1	FPI	POO2	POO2	IA	POO1
Horário 2	FPII	FPI	POO1	IA	FPII

5.2.3. Avaliação

Cada cromossomo é avaliado segundo o grau de atendimento aos requisitos desejáveis. Cada requisito desejável apresenta um peso, que será utilizado como valor em pontos para aqueles cromossomos que consigam cumprir estes requisitos. Todo cromossomo tem uma pontuação, resultado de uma avaliação sobre o atendimento dos requisitos desejáveis.

Tendem a sobreviver aqueles indivíduos com pontuação mais alta, ou seja, aqueles indivíduos mais adaptados ao meio. Ao longo da execução do sistema, sempre existirá um cromossomo que tenha pontuação mais alta e que seja considerado como solução, até que outro cromossomo, com pontuação mais alta, seja criado.

O cálculo do peso para cada requisito desejável segue os seguintes critérios:

- A opção do professor por concentrar suas aulas, avaliada da seguinte maneira:

  Peso = (Número de períodos lecionados) – (Número de dias lecionados)

- A opção do professor por espalhar suas aulas, avaliada da seguinte maneira:

  Peso = (Número de dias lecionados) – (Mínimo de dias possíveis de serem lecionados)

- O fato do professor não lecionar dois horários seguidos para a mesma turma. A princípio, são atribuídos +10 pontos ao peso de cada cromossomo, considerando que não existem professores lecionando em dois períodos seguidos para a mesma turma. A seguir, a cada dupla de horários que infrigir esta regra, haverá uma penalidade de –2 pontos no *fitness* daquele cromossomo.

- Disponibilidade de professores: a cada restrição atendida de cada professor é atribuído peso 5. Essa restrição pode ser modelada como:

$$\sum_{p=1}^{prof} \sum_{j=1}^{restr} 5\, r_{p\ j} \qquad (5.1)$$

sendo $r_{pj} = 1$ se a restrição j do professor i foi atendida e $r_{pj} = 0$, em caso contrário.

O *fitness* de um cromossomo é dado pela seguinte correlação matemática:

$$fit = \sum_{p=1}^{prof} (C_P (PL_P - DL_P) + (-1)(C_P - 1)(DL_P - MinDL_P) + \sum_{j=1}^{rest} 5R_{PJ}) + \sum_{i=1}^{turmas} \sum_{h=1}^{periodos/2} (-2P_{p_i h} P_{p_{i(h+1)}}) + 10 \quad (5.2)$$

Nessa expressão, tem-se que:

- $P_{pih} = 1$, se o professor p leciona no horário h para a turma i, e $P_{pih} = 0$, em caso contrário;

- $C_p = 1$, se o professor p optou por concentrar suas aulas e $C_p = 0$, em caso contrário;

- PL_p : total de períodos lecionados pelo professor *p* durante a semana;
- DL_p : total de dias lecionados pelo professor *p* durante a semana;
- $MinDL_p$: numero mínimo de dias possíveis de serem lecionados pelo professor *p*, com base em sua carga horária semanal.

5.2.4. Mutação

A mutação sobre um quadro de horários é uma operação que muda as características do quadro de horários inicial, de forma a manter certa similaridade, assim como uma mutação de um cromossomo altera algumas de suas características. A mutação consiste na troca de ordem das disciplinas das turmas entre os períodos selecionados para participar da referida operação.

O problema consiste em identificar um número *M* de períodos entre os 10 períodos existentes que devem participar da mutação. Este valor não deve ser muito grande, para não perder a similaridade com o quadro inicial. Também não pode ser muito pequeno, pois dificultará o processo de mutação, podendo levar à ocorrência de situações em que esta operação gera um quadro idêntico ao original, sem que a mesma tenha qualquer efeito na solução do problema. O cromossomo mutante gerado será considerado um novo cromossomo, estando sujeito às mesmas regras de sobrevivência dos demais cromossomos. A tabela 5.3 apresenta um cromossomo que vai sofrer mutação, com os campos sorteados para permutação. A tabela 5.4 mostra este mesmo cromossomo, depois da mutação.

Tabela 5.3 - Horário que sofrerá mutação, com *M* períodos selecionados.

	Segunda	Terça	Quarta	Quinta	Sexta
Horário 1	FPI	POO2	POO2	IA	POO1
Horário 2	FPII	FPI	POO1	IA	FPII

Tabela 5.4 - Horário "mutante"

	Segunda	Terça	Quarta	Quinta	Sexta
Horário 1	FPI	POO1	POO2	IA	POO1
Horário 2	FPII	FPI	IA	POO2	FPII

5.2.5. Operação de *Crossover*

A operação de *crossover* consiste em gerar novos cromossomos a partir de dois cromossomos originais, chamados pai e mãe. Estes novos elementos devem manter as características dos pais.

Um quadro de horários filho é gerado a partir de horários distribuídos no quadro pai e no quadro mãe. Para isso, é necessário o conceito de ciclo.

Dado um conjunto de turmas t e um conjunto de períodos p, um ciclo para t turmas gerado a partir de p períodos é um subconjunto mínimo de períodos do quadro de horários, contendo p períodos, tal que as disciplinas d_i, pertencentes à turma t_i do conjunto t nos períodos do ciclo são as mesmas para ambos, pai e mãe. Isto vai permitir a geração de dois elementos: aquele que contém as disciplinas do ciclo do pai e demais disciplinas da mãe, e aquele que contém as disciplinas do ciclo da mãe e as demais do pai.

A tabela 5.5 e a tabela 5.6 apresentam os elementos selecionados para a operação de *crossover.* Está em destaque um ciclo comum nos dois quadros. A tabela 5.7 apresenta o filho que recebeu o ciclo do pai e demais disciplinas da mãe e a tabela 5.8 apresenta o outro filho, que recebeu o ciclo da mãe e demais disciplinas do pai.

Tabela 5.5 - Horário pai com o ciclo

	Segunda	Terça	Quarta	Quinta	Sexta
Horário 1	FPI	POO1	POO2	IA	POO1
Horário 2	FPII	FPI	IA	POO2	FPII

Tabela 5.6 - Horário mãe com o ciclo

	Segunda	Terça	Quarta	Quinta	Sexta
Horário 1	POO1	IA	FPII	FPI	IA
Horário 2	FPI	FPII	POO1	POO2	POO2

Tabela 5.7 - Filho 1

	Segunda	Terça	Quarta	Quinta	Sexta
Horário 1	FPI	POO1	FPII	IA	IA
Horário 2	FPII	FPI	POO1	POO2	POO2

Tabela 5.8 - Filho 2

	Segunda	Terça	Quarta	Quinta	Sexta
Horário 1	POO1	IA	POO2	FPI	POO1
Horário 2	FPI	FPII	IA	POO2	FPII

5.2.6. Modelos de Algoritmos Genéticos Paralelos

Como o paralelismo é uma característica intrínseca de algoritmos genéticos, muitas implementações de algoritmos genéticos paralelos são encontradas na literatura [9]. Existem três modelos de implementação: modelo de população global, modelo de ilhas e modelo celular.

O primeiro modelo utiliza uma população simples, enquanto a avaliação dos indivíduos e a aplicação dos operadores genéticos são feitas em paralelo.

A segunda classe de algoritmos genéticos paralelos é desenvolvida sobre o modelo de ilhas. Neste modelo, indivíduos são divididos em subpopulações, em processos concorrentes, cada qual responsável por suas operações locais. Através de um processo de migração, os indivíduos de melhor avaliação podem migrar de uma ilha para outra.

Uma terceira classe de algoritmos genéticos paralelos utiliza o modelo celular, com o uso de um grande número de pequenas subpopulações. A população é então mapeada em um grafo conectado de processadores. O baixo custo de comunicação entre os processadores acarreta alta migração de indivíduos entre eles.

O modelo de ilhas foi adotado neste trabalho, devido, basicamente, a essa característica de custo computacional.

5.2.7. Arquitetura da Implementação

A base da implementação é a linguagem JAVA, distribuída por um *cluster* de computadores utilizando ambiente CORBA, com a plataforma Java IDL. O sistema é composto de:

- Interface de Usuário: aplicativo Java, que recebe os dados como cadastro das turmas, professores e períodos disponíveis. Executa no cliente. Recebe as soluções das populações e seleciona o cromossomo mais bem avaliado como solução temporária do problema;

- População: aplicativo Java, que contém um conjunto de cromossomos que estão sujeitos às operações de mutação, *crossover*, avaliação e

eliminação, caso não tenha uma boa avaliação. Cada população é executada em uma das máquinas do *cluster*.

A operação de *crossover* pode acontecer entre elementos de populações diferentes. Para que isto aconteça, os cromossomos que representam as melhores soluções são registrados no servidor de nomes CORBA. Cada população escolhe elementos registrados no servidor de nomes para realizar operação de *crossover* com seus elementos mais bem avaliados. A figura 5.1 mostra os elementos presentes no sistema.

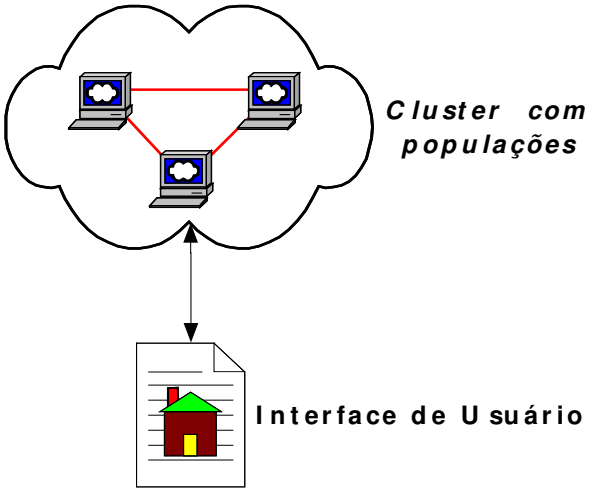


Figura 5.1 - Elementos do sistema

5.2.8. Interface CORBA

A interface CORBA utilizada, escrita em IDL (*Interface Definition Language*), deve permitir a interação dos diversos aplicativos existentes nesta implementação.

```
module IHorario {

  interface IDisciplina {
    long getProfessor();
    string getCodigo();
    long getCreditos();
  };

  typedef sequence<IDisciplina> Periodo;
  typedef sequence<boolean> VBool;
  typedef sequence<VBool> Restricao;

  interface ITurma {
    Periodo getDisciplinas();
```

```
  };long getNumDis();

  typedef sequence<ITurma> Curso;

  interface ICromossomo{
    long getProfessor(in long p, in long t);
    void setProfessor(in long p, in long t, in long prof);
    long getPeriodo(in long prof, in long t, in long p0);
    ICromossomo filhos(in ICromossomo pai);

    void setMatriz(in long t, in long prf, in long p, in boolean valor);
    boolean getMatriz(in long t, in long prf, in long p);
    long getNumPeriodos();
    long getNumTurmas();
  };

  interface IRequisito {
    long getPeso();
    void setPeso(in long peso);
    long getProfessor();
    void setProfessor(in long prof);
    boolean avalia(in ICromossomo Cr);
  };

  typedef sequence<IRequisito> Requisitos;

  interface IQuadro {
    Curso getTurmas();
    long getNumTurmas();
    long getNumProfessores();
    long getNumPeriodos();
    long put Fitness(in long pt);
    Requisitos getRequisitos();
    Restricao getRestricaoDeve();
    Restricao getRestricaoNaoPode();
  };
};
```

Figura 5.2 - Interface de comunicação entre os módulos.

5.2.9. Modelagem do Sistema

Um algoritmo genético pode ser escrito com base no seguinte pseudocódigo:

```
1.  t = 0
2.  gerar população inicial P(0)
3.  Avaliar P(0)
4.  enquanto critério de parada não for atendido faça
5.  t = t + 1
6.  selecionar população P(t) à partir de P( t -1)
7.  aplicar operadores de cruzamento sobre P(t)
8.  aplicar operadores de mutação sobre P(t)
```

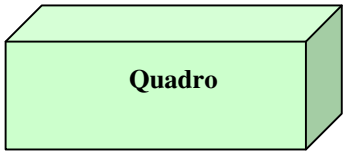
9. avaliar P(t)
10. voltar ao passo 4

Figura 5.3 – Pseudocódigo de algoritmo genético.

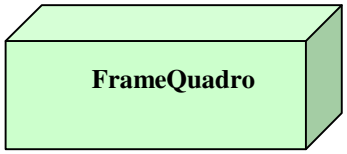
O sistema foi desenvolvido com base no contexto da PPOO (programação paralela orientada a objetos) e tem as seguintes classes implementadas:

a) Classe Quadro: inicializa um módulo do sistema. Faz a montagem do frame e invoca o FrameQuadro.

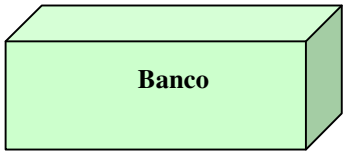
.



b) Classe FrameQuadro: sua finalidade é fazer uma interface com o usuário, de modo a prover a instância do problema - visualização dos dados e interação entre eles - o *start* do sistema e o pedido de mostra da melhor solução até aquele momento. Seus atributos configuram toda a situação do problema, tais como: professores, turmas, disciplinas de cada turma, restrições e requisitos de professores. Tais informações são fomentadas pela classe Banco.

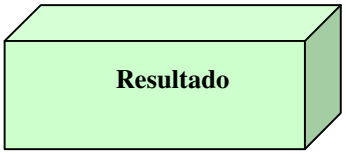


c) Classe Banco: faz a interface entre o banco de dados fomentador dos dados da aplicação e o sistema em si. Possui vários scripts SQL com a finalidade de suprir todas as informações demandadas pelo FrameQuadro.

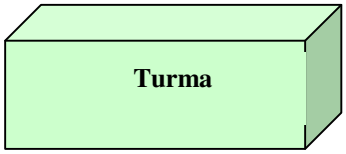


d) Classe Resultado: é invocado pelo FrameQuadro, que passa o cromossomo representando o indivíduo mais bem avaliado até então. É

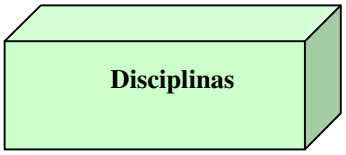
responsável pela montagem do frame do quadro de horário, ou seja, plota a solução em um dispositivo de saída de dados.



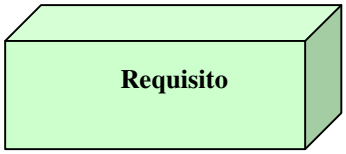
e) Classe Turma: instancia informações sobre as turmas envolvidas no horário, tais como seu código e lista das disciplinas de cada turma. Possui interface, ou seja, é uma única classe invocada por todas as populações.



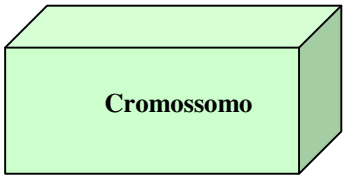
f) Classe Disciplinas: instancia informações a respeito de cada disciplina envolvida no quadro de horários tais como o professor que a ministra em uma determinada turma e seus créditos. Também possui uma interface.



g) Classe Requisito: responsável por gerenciar os requisitos de um professor. Aqui requisitos são vistos como a opção de um professor *p* em concentrar ou não suas aulas durante a semana. Esta classe avalia e registra o cálculo do *fitness* em um de seus atributos para estas restrições. É uma classe interfaceada.



h) Classe Cromossomo: um cromossomo é um indivíduo em uma população, dentro do contexto de algoritmos genéticos. Portanto, um indivíduo é um quadro de horários válido ou não. Um horário configura-se sob vários aspectos, que compõem suas características e propriedades. A despeito deste conceito, a presente classe apresenta os seguintes atributos:

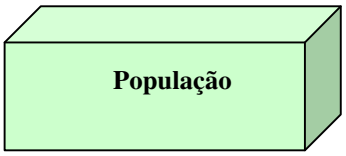


- Quantidade de professores, turmas e períodos envolvidos no horário;
- Uma matriz tridimensional, que é o quadro propriamente dito, ou seja, uma matriz de valores lógicos, que informa se um professor *p* leciona em uma turma *t* no período *p*;
- Uma classe requisito;
- A lista de restrições com relação à disponibilidade de cada professor;
- A lista de restrições com relação às necessidades administrativas e/ou pedagógicas (relevância de um professor *p* lecionar em um período *p*) de cada professor;
- Seu *fitness*.

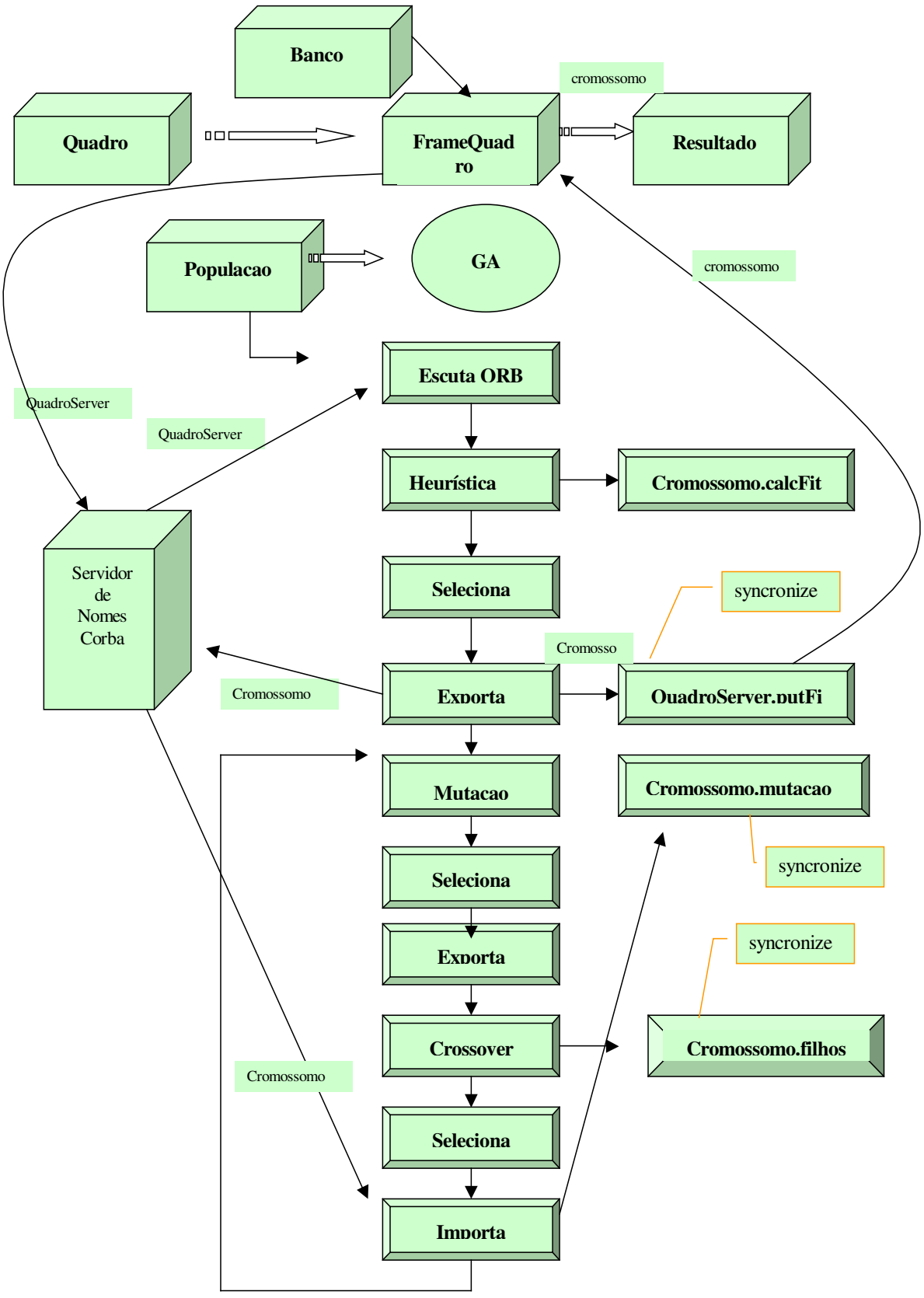
A classe cromossomo habilita-se a:

- Calcular seu *fitness*;
- Verificar se há sobreposição de professor e/ou turma naquele indivíduo;
- Fazer uma cópia de si mesmo;
- Executar operações genéticas de mutação;
- Executar operações genéticas de *crossover* entre um cromossomo pai e um cromossomo mãe, retornando dois filhos;
- Implementar um semáforo para aqueles cromossomos que estão sob operações genéticas.

i) Classe População: implementa um algoritmo genético propriamente dito. Nela, é gerada uma população inicial de tamanho *N*, através de uma heurística construtiva e, a partir deste ponto, o processo evolutivo é iniciado, através do princípio de evolução e seleção das espécies. Cabe ressaltar que esta classe é tratada como uma *thread*, ou seja, é possível processar várias populações em servidores distintos ou não. A exportação e importação dos indivíduos mais bem adaptados entre populações são feitas de forma que haja fortalecimento e dispersão das características dos indivíduos de cada uma delas.



5.2.9.1. Diagramação do sistema



5.2.9.2. Descrição do sistema

Três aplicações compõem o sistema como um todo. A primeira é o servidor de nomes, responsável por listar o nome e a referência de cada objeto compartilhado por toda a aplicação. As duas outras aplicações são as classes *Quadro* e *População*.

Um único servidor deve executar o servidor de nomes assim como a classe *Quadro*. Já a classe *População* pode ser executada em vários servidores. Não há restrição quanto a um mesmo servidor estar executando um subconjunto destas três aplicações.

A classe *População* fica escutando um registro no servidor de nomes e, a partir, daí inicia seus processos. O primeiro registro no servidor de nomes é de responsabilidade da classe *FrameQuadro*. Isto ocorre quando o usuário ordena o início de todo o processo de geração do quadro de horários.

Os *inputs* são fornecidos por um banco de dados, em que estão mapeados todos os detalhes do problema. Estes *inputs* são fomentados ao *FrameQuadro* automaticamente durante sua instanciação. A qualquer tempo após a geração da população inicial (heurística construtiva), é possível visualizar a melhor solução encontrada até então. Também esta operação é feita através da classe *FrameQuadro*.

Mas é na execução da classe *População* que todo o processo inteligente do sistema ocorre. O ciclo é iniciado através da geração aleatória de uma população inicial, que é gerada através de força bruta, em que se tenta alocar professores a turmas em horários aleatórios, descartando sempre soluções infactíveis, mas não se preocupando com o atendimento às restrições desejáveis do problema. Após esta etapa um ciclo de gerações populacionais é processado. Cada geração populacional sofre um conjunto de operações genéticas. Em um primeiro momento, os indivíduos da população sofrem mutação, são classificados e selecionados para, logo em seguida, os melhores indivíduos serem registrados no servidor de nomes, de forma que outras populações possam incluí-los como um de seus indivíduos. Numa etapa posterior, há uma operação de *crossover* entre pares de cromossomos, advindo daí dois novos filhos que farão parte daquela população. Novamente, há uma classificação e exportação dos melhores indivíduos. Ao final do ciclo, aquela população recebe um par de indivíduos

registrados por outras populações e os cruza, gerando mais dois novos indivíduos, que também passarão a integrar a mesma.

A cada processo de exportação de cada população, ocorre uma verificação a respeito da geração ou não de uma solução melhor que aquela tida como melhor até aquele instante *t*. Sempre que isto ocorrer, o servidor de quadros instanciará aquele quadro como sendo o ótimo. É esta classe a responsável por repassar este mesmo cromossomo à *FrameQuadro* quando da invocação da mostra do melhor quadro.

Um mesmo cromossomo não pode sofrer operações genéticas por mais de uma população, paralelamente. Desta forma, toda operação genética sobre um cromossomo dever ser gerenciada por um semáforo.

O ponto de parada é o alcance da satisfação do resultado obtido ou o alcance do ótimo global. Para isto, obviamente, é necessário um cálculo prévio do melhor *fitness* possível para aquela instância do problema.

CAPÍTULO VI EXPERIMENTOS E RESULTADOS COMPUTACIONAIS

Foram criadas três instâncias do problema para efeitos de testes. O problema de quadro de horários se particulariza em cada instituição considerada. No entanto, é importante observar que existem características que são comuns a diversas instituições, de modo que se possa definir as diversas modalidades desse problema, como apresentadas no capítulo 4.

Como o propósito deste trabalho foi adotar, como estudo de caso, o projeto pedagógico e as diretrizes dos cursos da área de computação sediados no campus Lafaiete (FATEC) da Universidade Presidente Antônio Carlos – UNIPAC, não foi possível fazer uma avaliação e comparação com resultados científicos de trabalhos correlatos, pelo fato de cada um destes trabalharem dentro de contextos distintos que definem a instância do problema. Desta forma, então, procurou-se avaliar a eficácia da solução assim como sua eficiência computacional. Acrescente-se a isto que, a partir deste trabalho, será possível, em projetos futuros, aplicar, à abordagem metaheurística de algoritmos genéticos, uma avaliação do emprego do paralelismo computacional.

Todas as instâncias dos problemas abordados foram baseadas em situações reais, vivenciadas na instituição supracitada. A metodologia adotada foi a abordagem do caso mais simplificado a uma situação mais crítica do problema.

A rede de computadores utilizada consistiu de três computadores PC, padrão Intel, com as seguintes configurações:

- processador K6/2 450 com 128 MB de memória RAM;
- *notebook* Celeron 2.4 com 512 MB de memória RAM;

- Sempron 2.2 com 256 MB de memória RAM.

6.1. Instâncias do problema

6.1.1. Teste 1

Nesta instância do problema procurou-se pontuar um caso mais simples sem, no entanto, perder as particularidades da instituição objeto de estudo. Considerou-se um subconjunto de 11 professores e 2 turmas do curso de Sistemas de Informação. A restrição de distribuição de aulas foi dada a 8 professores e, aos 3 restantes, foi atribuída a restrição de concentração de aulas no decorrer da semana. Foram trabalhadas 16 restrições de indisponibilidades de horários, distribuídas entre 4 professores.

A instância deste problema mapeou, contudo, um caso real ocorrido dentro do curso considerado sem, no entanto, considerar o universo do problema e sim apenas uma subconjunto do mesmo.

O *fitness* da solução ótima é calculado com base nos seguintes elementos da instância trabalhada :

- com relação às restrições de indisponibilidades: como, para cada restrição de indisponibilidade atendida, é atribuído +5 pontos à solução, tem-se o seguinte pré cálculo : 16 restrições * 5 pontos = 80. Portanto, um indivíduo que atende a todas as restrições de indisponibilidade terá 80 pontos com relação a esta restrição nesta instância.

- com relação a restrição de concentrar aulas: para esta restrição, como descrito na seção 5.2.3, tem-se o seguinte cálculo:

  Peso = períodos lecionados – número de dias trabalhados

Considerando-se, então, o número de períodos selecionados entre os 3 professores optantes desta restrição, conclui-se que o indivíduo que tenha atendido à restrição dos 3 professores em questão terá +3 pontos atribuído ao seu *fitness* para esta instância.

- com relação a restrição de distribuir aulas: para esta restrição, como descrito na seção 5.2.3, tem-se o seguinte cálculo:

  Peso = número de dias trabalhados - mínimo de dias possíveis

  Considerando então o número mínimo de dias possíveis de serem trabalhados entre cada um dos 8 professores optantes desta restrição, tem-se que, a cada solução que atenda estas restrições para todos os professores em questão, será atribuído + 5 pontos.

- com relação a professor não lecionar por 2 períodos sucessivos para a mesma turma $\Rightarrow$ 2*10 = 20. Portanto, àquela solução que não infrinja esta restrição em nenhuma turma, tem-se + 20 pontos em seu *fitness*.

No entanto há um professor com pedido de concentração de aulas lecionando para uma única turma, o que, conseqüentemente, acarretará sua presença em 2 períodos sucessivos na mesma. Assim, a aplicação não deve concentrá-lo, o que irá acarretar uma perda de 1 ponto no *fitness* daquela solução.

Com base na expressão do cálculo do *fitness* de um indivíduo, dado por :

$$fit = \sum_{p=1}^{prof} (C_P (PL_p - DL_p) + (-1)(C_P - 1)(DL_p - MinDL_p) + \sum_{j=1}^{rest} 5R_{PJ}) + \sum_{i=1}^{turmas} \sum_{h=1}^{periodos-2} (-2P_{p_{ih}} P_{p_{i(h+1)}}) + 10$$

O cálculo do *fitness* ótimo, nesta instância, será dado por :

Total = 80 + 3 + 5 + 20 – 1 = 107

ou seja, o valor do fitness ótimo será dado por 107.

6.1.1.1. Resultados

Para esta instância do problema, foram feitas 10 execuções do algoritmo, a fim de se aferir a média das melhores soluções alcançadas, assim como o tempo médio de alcance da melhor solução. O critério de parada utilizado foi o alcance da solução ótima ou a observação da estabilização na melhora das soluções.

A descrição aqui feita vale para as instâncias dos testes 2 e 3.

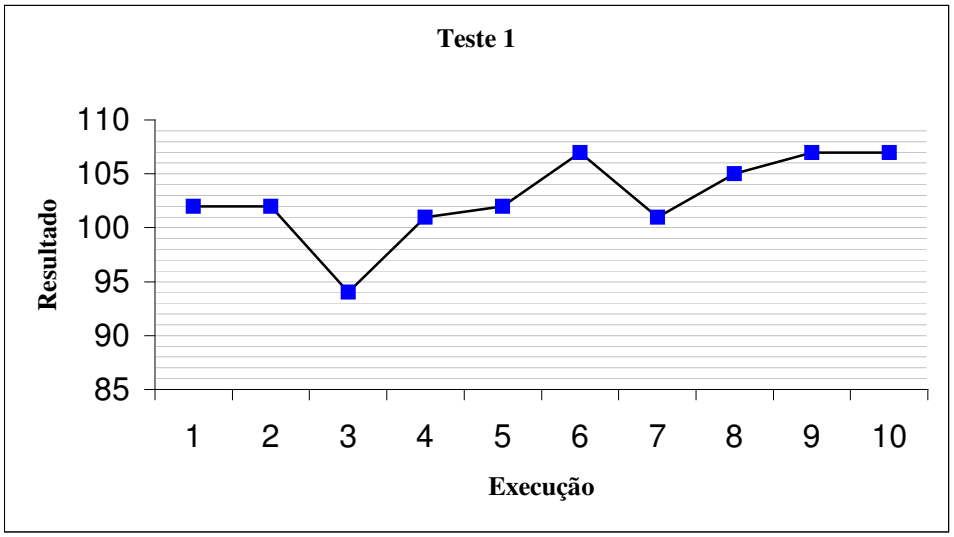


Gráfico 6.1. Teste 1

Nesta bateria de testes houve um *fitness* médio igual a 102,8 (~96% do ótimo). A solução ótima foi alcançada por 3 vezes em um tempo médio de 8,7 segundos.

6.1.2. Teste 2

Nesta instância do problema, abordou-se um caso um pouco mais complexo dentro do contexto da instituição. Considerou-se um subconjunto de 11 professores e 2 turmas do curso de Sistemas de Informação. A restrição de distribuição de aulas foi dada a 8 professores e, aos 3 restantes, foi atribuída a restrição de concentração de aulas no decorrer da semana. Foram trabalhadas, no entanto, 66 restrições de indisponibilidades de horários, distribuídas entre 10 professores.

Com base no mesmo raciocínio exposto no teste 1, os cálculos de atribuição de pesos para cada restrição considerada, a fim de se aferir o *fitness* da solução ótima, são os seguintes:

- com relação às restrições de indisponibilidades :

  66 restrições * 5 pontos = 330.

  Portanto, um indivíduo que atende todas as restrições de indisponibilidades terá 330 pontos com relação a esta restrição nesta instância.

- com relação àa restrição de concentrar aulas: considerando-se o número de períodos selecionados entre os 3 professores optantes desta restrição, conclui-se que o indivíduo que tenha atendido à restrição dos 3 professores em questão terá + 3 pontos atribuído ao teu *fitness* para esta instância.

- com relação a restrição de distribuir aulas: considerando o número mínimo de dias possíveis de serem trabalhados entre cada um dos 8 professores optantes desta restrição, tem-se que a cada solução que atenda estas restrições para todos os professores em questão será atribuído + 5 pontos.

- com relação a professor não lecionar por 2 períodos sucessivos para a mesma turma $\Rightarrow$ 2*10 = 20. Portanto, àquela solução que não infrinja esta restrição em nenhuma turma, tem + 20 pontos em seu *fitness*.

No entanto há um professor com pedido de concentração de aulas lecionando para uma única turma, o que, conseqüentemente, acarretará sua presença em 2 períodos sucessivos na mesma. Assim, a aplicação não deve concentrá-lo, o que irá acarretar uma perda de 1 ponto no *fitness* daquela solução. Sendo assim, o calculo do *fitness* ótimo, nesta instância, é dado por:

$$\text{Total} = 330 + 3 + 5 + 20 - 1 = 357$$

e, portanto, o fitness ótimo para essa instância será dado por 357.

6.1.2.1. Resultados

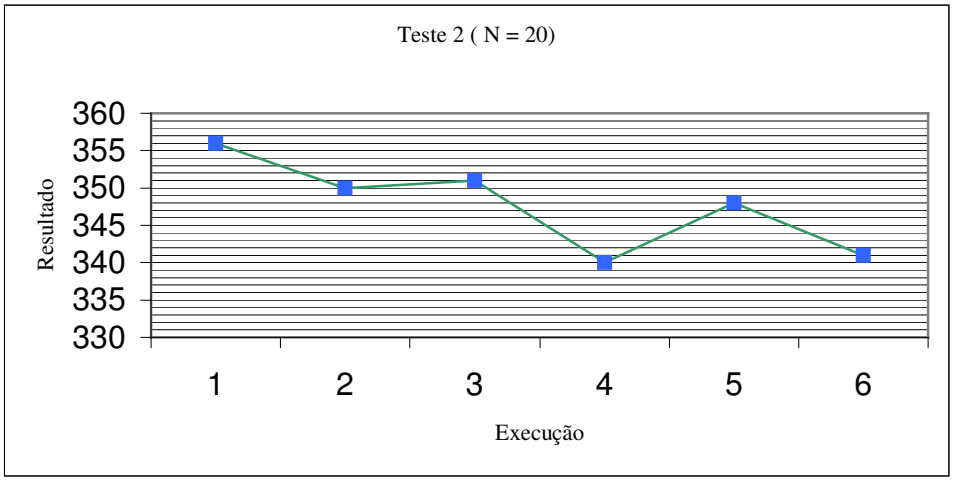


Gráfico 6.2. Teste 2

Com uma população de 20 indivíduos, obteve-se um *fitness* médio igual a 347,7 (~97% do ótimo). O tempo médio de alcance dos valores máximos para esta bateria girou em torno de 9,6 segundos.

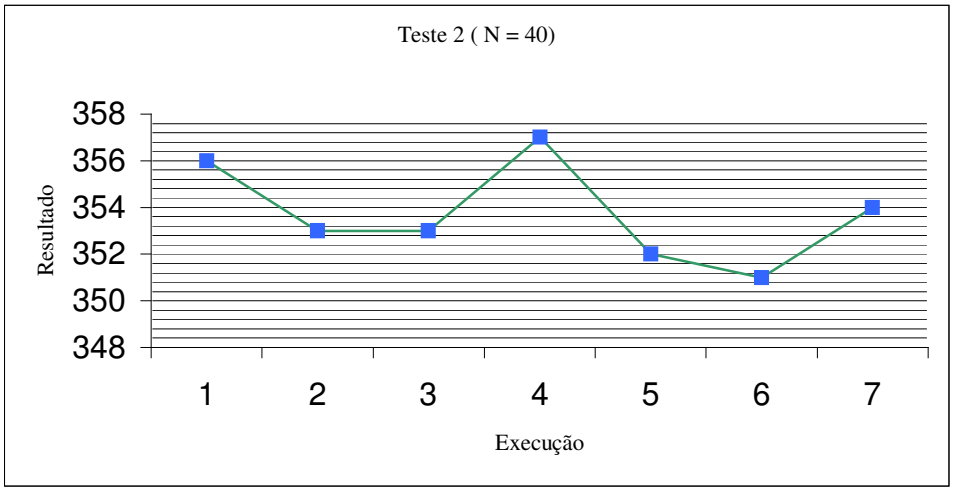


Gráfico 6.3. Teste 2

Dobrou-se o tamanho da população e o *fitness* médio passou a ser igual a 353,7 (~99,07% do ótimo). Alcançou-se o ótimo uma única vez em 11,4 segundos.

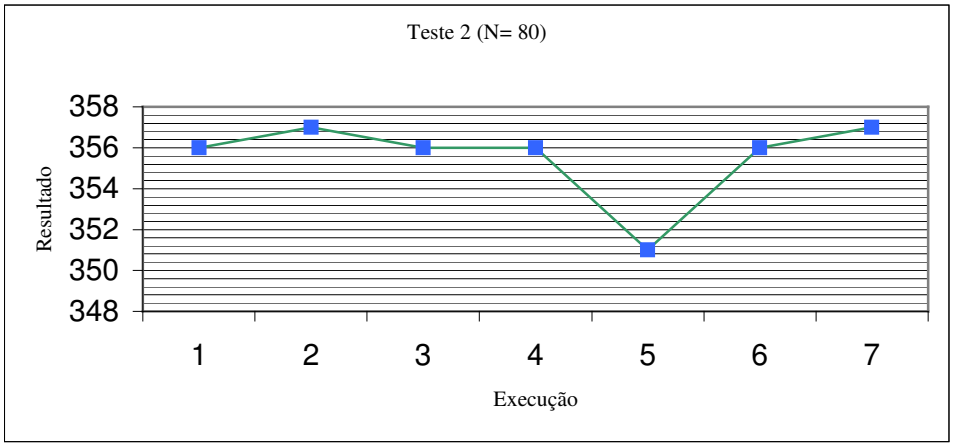


Gráfico 6.4. Teste 2

O *fitness* médio para uma população tamanho 80 sofreu uma leve melhora, passando para 355,6 (~99,6% do ótimo). Alcançou-se o ótimo por 2 vezes e um tempo médio de 15,7 segundos.

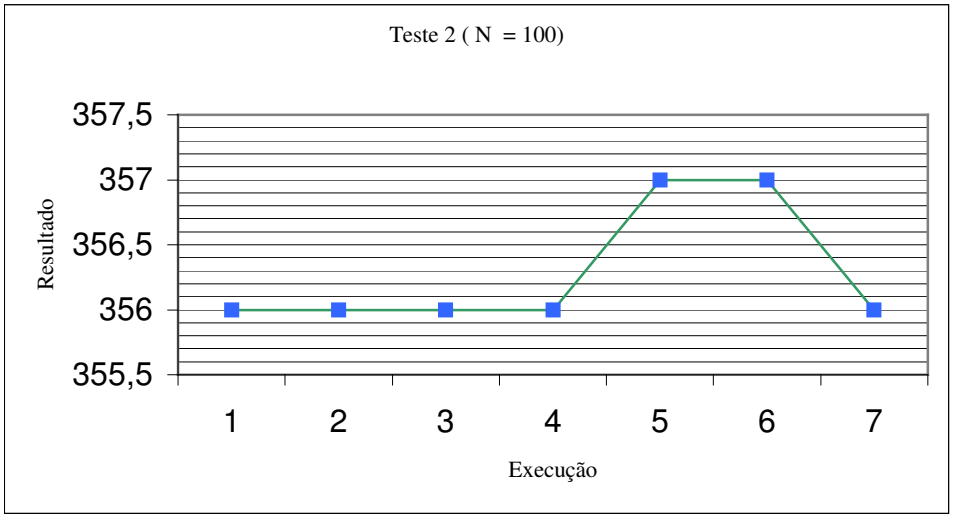


Gráfico 6.5. Teste 2

Com 100 indivíduos por população o *fitness* médio passou a 356,3 (~99,8% do ótimo) sendo que o alcance do ótimo ocorreu em uma média de 18,05 segundos.

6.1.3. Teste 3

Esta instância do problema mapeou o universo de uma situação real ocorrida no início do ano de 2005. Considerou-se um conjunto de 25 professores e todas as turmas do curso de Sistemas de Informação, ou seja, 8 turmas. A restrição de distribuição de aulas foi dada a 13 professores e, aos 12 restantes, foi atribuída a restrição de concentração de aulas no decorrer da semana. Foram trabalhadas 132 restrições de indisponibilidades de horários, distribuídas entre todos os professores do curso.

Desta forma, o cálculo do *fitness* ótimo, nesta instância, é dado por:

$$\text{Total} = 660 + 35 + 80 = 775$$

de modo que o fitness ótimo da instância é dado por 775.

Como a instância do problema considerada neste teste é a de uma situação real e, portanto, possui maior alta complexidade, aferiu-se também o comportamento da aplicação utilizando os recursos de paralelismo e a mesma situação sem o paralelismo. Uma comparação entre o tempo de busca nas duas situações é mostrado como forma de pontuar as melhorias propostas por este trabalho para a técnica de algoritmos genéticos através do paralelismo computacional.

6.1.3.1. Resultados

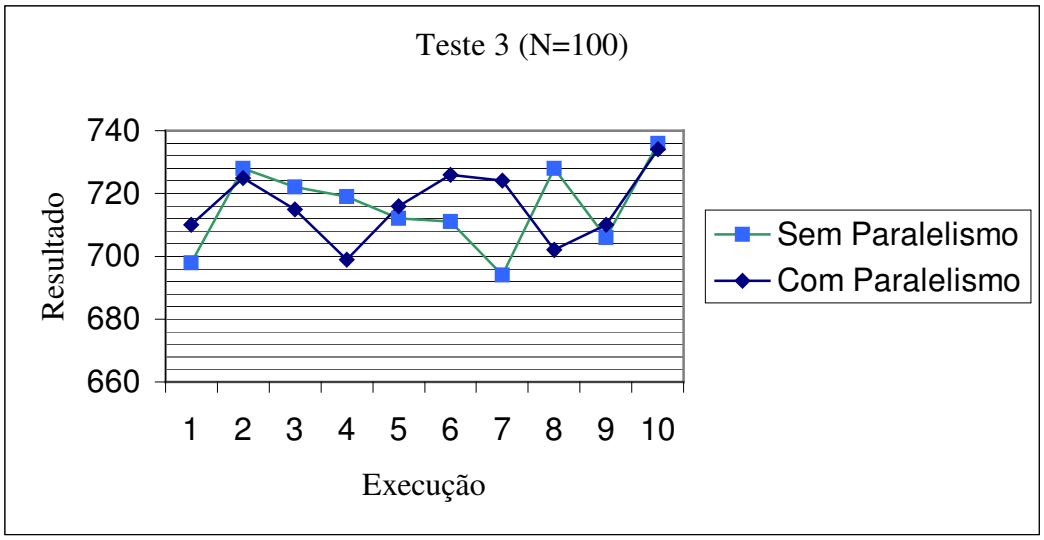


Gráfico 6.6. Teste 3

Com a aplicação utilizando o paralelismo computacional, o *fitness* médio foi de 715,4 (~ 92,3% do ótimo). Como se pode observar no gráfico, não houve alcance do ótimo e o tempo de alcance da melhor solução foi de 78 segundos. Após isto, o algoritmo não mais conseguiu convergir. Na aplicação *stand alone*, houve um *fitness* médio de 716,1 (~ 92,4% do ótimo), com um tempo médio de alcance da melhor solução da ordem de 140 segundos.

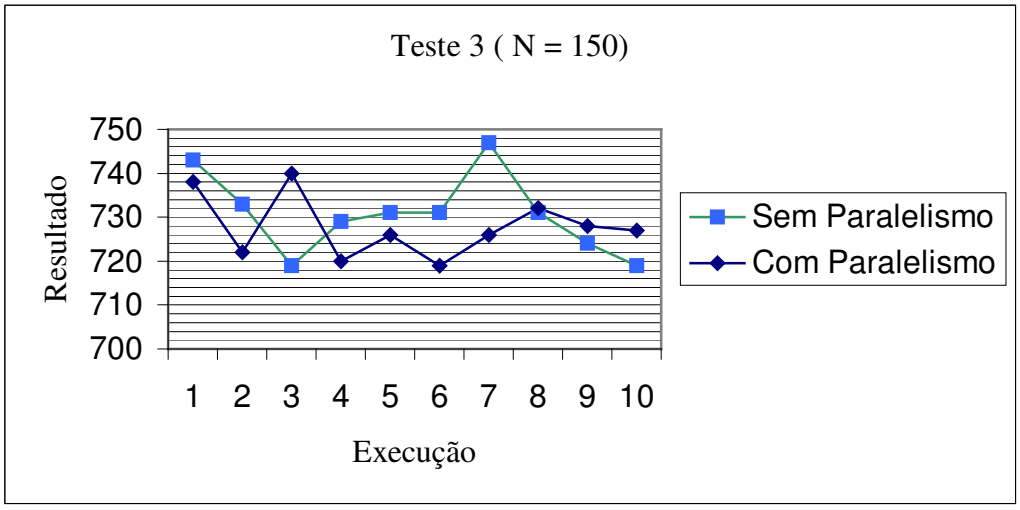


Gráfico 6.7. Teste 3

Com o aumento da população, houve uma sensível melhora do *fitness* médio em ambas as aplicações, elevando-se para 730,7 (~ 94,2% do ótimo) e 727,8 (~93,9% do ótimo) para a aplicação paralela e *stand alone*, respectivamente. Também não

houve alcance da melhor solução. A maior pontuação nesta bateria levou 126 e 192 segundos para ser alcançada na aplicação parelela e *stand alone*, respectivamente.

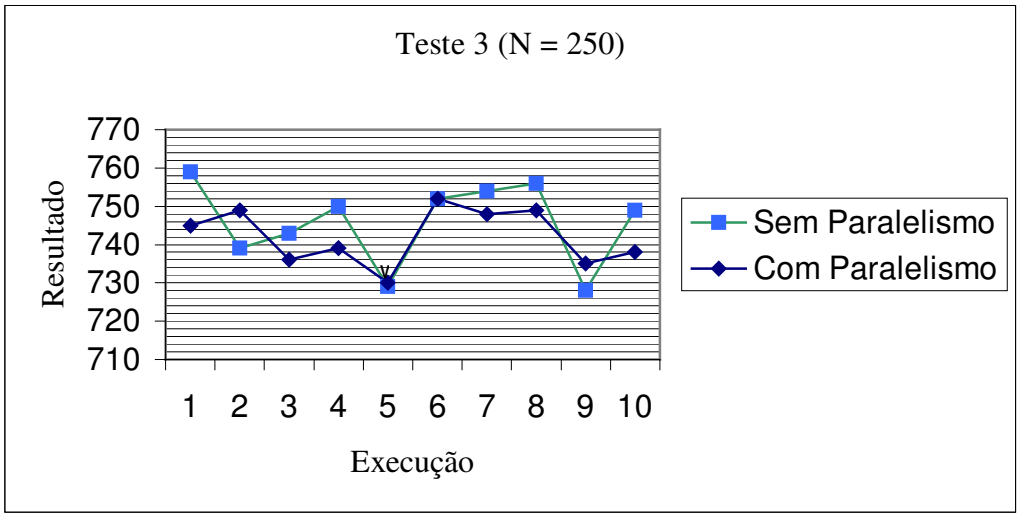


Gráfico 6.8. Teste 3

Com um *fitness* médio de 745,9 (~96,2% do ótimo) e 742,1 (~95,8% do ótimo) no paralelismo e sem o paralelismo, respectivamente, e um tempo de alcance da melhor pontuação de 188 e 266 segundos (paralelismo/sem paralelismo), não houve alcance do ótimo.

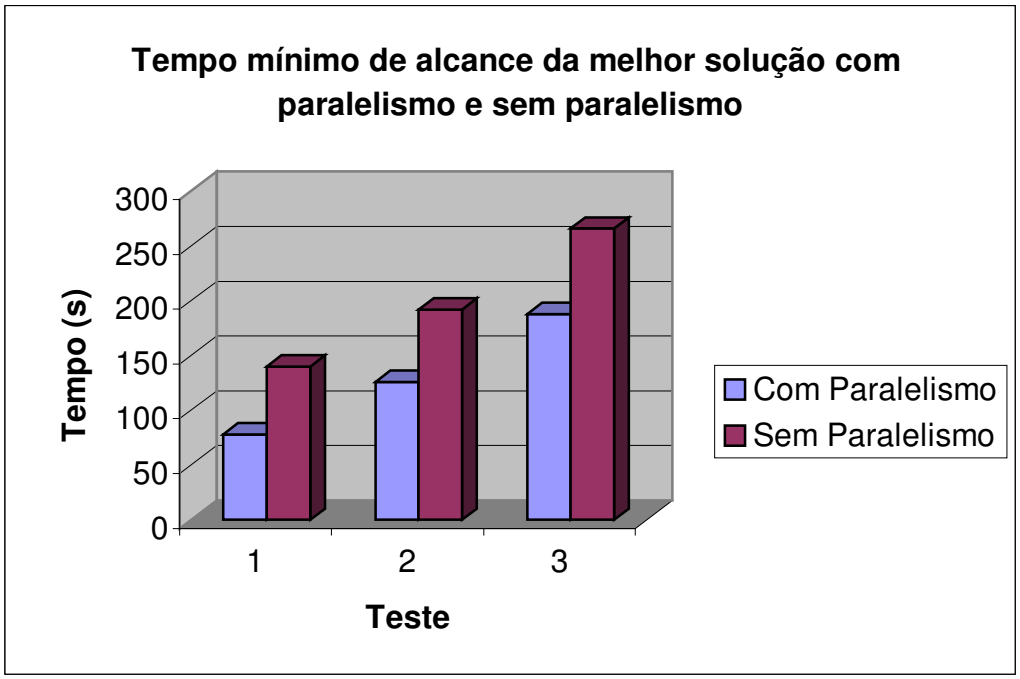


Gráfico 6.9. Teste 3

Pelos gráficos 6.6, 6.7 e 6.8 pode-se concluir que o comportamento das curvas (alcance da melhor solução – parelelismo X *stand alone*) em relação ao alcance da melhor solução são bastante similares. Portanto, o paralelismo não garante uma melhor solução quando comparado a uma aplicação que não utiliza paralelismo computacional.

O gráfico 6.9 mostra, no entanto, a eficiência computacional do paralelismo. Houve um ganho médio com relação ao tempo de alcance da melhor solução nos 3 testes da ordem de 36% quando se recorria ao paralelismo computacional em detrimento da aplicação sem paralelismo.

CAPÍTULO VII CONCLUSÕES FINAIS E TRABALHOS FUTUROS

7.1. Conclusões Finais

Como não há, na literatura, pelo menos do conhecimento do autor dessa dissertação, um estudo sobre o quadro de horários escolar segundo os regimentos pedagógicos e administrativos da instituição objeto de estudo, fica difícil fazer uma comparação com trabalhos anteriores. No entanto, com base na confecção manual dos quadros de horários que já ocorreram na instituição objeto de análise, o algoritmo mostrou eficiência com relação a tempo e qualidade de confecção em todos os casos de teste. Seu comportamento demonstra que, à medida que o tamanho da população é ampliado, há melhorias em seus resultados e, obviamente, um custo computacional maior é acarretado. No entanto, isto já é uma característica peculiar de algoritmos genéticos e a abordagem do paralelismo computacional é justamente para atenuar custos que tornassem inviáveis a aplicação desta técnica para problemas desta categoria.

Verifica-se que a convergência para um ótimo ocorre de forma muito rápida e, a partir de certo ponto, não há melhorias, independentemente do tempo de execução, ou seja, a aplicação não mais tende a convergir para uma solução melhor. Portanto, melhorias no algoritmo em sua execução são passíveis de serem implementadas.

Outro problema encontrado situa-se na operação de *crossover*. Trata-se de um operador de alta complexidade na sua implementação. Constatou-se nesta aplicação que há a possibilidade de haver sobreposição de turmas (mesmo professor lecionando para 2 turmas, concomitantemente) quando da ocorrência de uma operação de

crossover entre dois cromossomos. Uma saída analisada e não implantada seria a implementação de um algoritmo como o *backtracking*, por exemplo, para evitar tal incongruência.

De todo, pode-se concluir que algoritmos genéticos tendem a ser eficientes para problemas de quadro de horários, ainda que a implementação dos operadores genéticos seja de alta complexidade.

A abordagem da computação paralela mostrou-se extremante eficaz, dada a questão da rapidez com que as convergências ocorrem, devido, principalmente, à troca de cromossomos entre as populações, o que fortalece a heterogeneidade das mesmas, assim como a obtenção de indivíduos cada vez mais adaptados. No entanto, o paralelismo computacional invoca uma série de sintonias que envolvem ambiente operacional, padrões de hardware, linguagens de programação, padrões de implementação, o que torna o ambiente de desenvolvimento altamente propício a erros.

7.2. Trabalhos Futuros

Algumas propostas de trabalhos futuros são apresentadas a seguir. São indicações ou de continuidade do relato aqui exposto ou novas implementações, não abordadas no desenvolvimento desse trabalho de pesquisa.

- A implementação do operador de *crossover* mostrou incongruências em algumas situações, acarretando sobreposição de turmas. Portanto, este operador é passível de melhorias, ficando latente sua complexidade, em decorrência da própria concepção da operação e, no caso da proposta do presente trabalho, ampliada pelo contexto da programação paralela;

- O grau de complexidade no uso de especificações CORBA é muito maior se comparado a outros padrões. Desta forma, abrem-se precedentes para implementação do paralelismo utilizando outros padrões, a fim de minorar tais dificuldades;

- Após algum tempo de convergência, verificou-se nos testes que o algoritmo não mais convergia, mesmo não tendo alcançado o ótimo global.

A questão é que, teoricamente, o método teria que garantir uma convergência, mesmo que de uma forma bastante lenta. A análise deste ponto pode vir a enriquecer ainda mais a técnica, elucidando características deficitárias dos algoritmos genéticos e propondo modificações, de forma a tornar a técnica totalmente adequada para a solução de quadro de horários;

- Paralelização dos cromossomos dentro da população. Isto pode permitir uma convergência mais rápida e utilização mais efetiva dos recursos requeridos pelo sistema;

- Abertura da especificação, permitindo a solução de um número maior de problemas deste tipo.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ABRAMSON, D.; ABELA, J. "A Parallel Genetic Algorithm for Solving the School Timetabling Problem". In Proceedings of the 15th Australian Computer Science Conference, Hobart, Australia, 1992.

[2] APPLEBY, J. S.; BLACK, D. V.; NEWMAN, E. A., "Techniques for Producting School Timetables on a Computer and Their Application to Other Scheduling Problems", The Computer Journal, v.3, pp. 237-245, 1960.

[3] BARCELLOS,J.C.H.. Algoritmos Genéticos Adaptativos: Um estudo Comparativo, 2000. Dissertação (Mestrado em Engenharia) – Escola Politécnica-Universidade do Estado de São Paulo. Disponível em < **http://www.teses.usp.br/teses/disponiveis/3/3141/tde-05092001-141334/publico/Tag.pdf** >. Acesso em 04/2005

[4] BARRACLOUGH, E. D., The Application of a Digital Computer to the Construction of Timetables, The Computer Journal, v.8, pp. 136-146, 1965.

[5] BERGHUIS, J., HEIDEN, A. J. van der, BAKKER, R., "The Preparation of School Time tables by Eletronic Computer", BIT, v.4, pp. 106-114, 1964.

[6] BOLTON, Fintan. Pure CORBA – A Code-Intensive Premium Reference. 1 ed. United States: Sams Publishing , 2002.

[7] BOND, Martin et al. , Aprenda J2EE com EJB, JSP, Servlets, JDBC, JDBS e XML em 21 dias. Tradução de João Eduardo Nóbrega Tortello. 1 ed. São Paulo: Makron Books , 2003.

[8] BRODER, S., "Final Examination Scheduling", Comm. A. C. M., v.7, pp. 494-498, 1964.

[9] BUBAK, M; SOWA, K – "Object Oriented Implementation of Parallel Genetic Algorithms"– in Buyya, Rajkumar – High Performance Cluster Computing, vol. 2, Prentice Hall, 1999

[10] CALDEIRA, J. C.; ROSA, A, C, School Timetabling using Genetic Search, PATAT (1997), pp. 115-122, 1997 .

[11] CARRASCO,M.P.; PATO,M.V..A Multiobjective Genetic Algorithm for the Class/Teacher Timetabling Problem. In : Third International Conference on Practice and Theory of Automated Timetabling 3, 2000, Konstanz, Germany. Anais do Third International Conference on Practice and Theory of Automated Timetabling III, London, UK, 2000. P. 3-17.

[12] CARVALHO,A.P.L.F. Algoritmos Genéticos. Disponível em < **http://www.icmc.usp.br/~andre/research/genetic/** >. Acesso em 03/2005.

[13] CERQUEIRA,M.E.; SANTOS,L.A.S.. Algoritmos Genéticos: Aspectos Matemáticos. Disponível em < **http://twiki.im.ufba.br/pub/DACOMP/SeminariosCientificos/lucasa.pps** >. Acesso em 05/2005.

[14] COLE, A. J., "The Preparation of Examination Timetables Using a Small-Store Computer", Comput. Journal., v.7, pp. 117-121, 1964.

[15] COLOMI, A.; DORIGO, M.; MANIEZZO, V. – "Genetic Algorithms: A New Approach to the Timet-Table Problem" - Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy

[16] COLOMI, A.; DORIGO, M; MANIEZZO, V – "Genetic Algorithms and Highly Constrained Problems: The Time-Table Case" – Proceedings of the First International Workshop on Parallel Problem Solving from Nature, Dortmund, Germany, Lecture Notes in Computer Science 496, Springer-Versalg, 55-59

[17] COOPER, T. B.; KINGSTON, J. H. – "The complexity of Timetable Construction Problems" – in Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT ´95, volume 1153 of Lecture Notes in Computer Sciences, 1995

[18] CSIMA, J., "Investigations on a Time-Table Problem. Ph.D. thesis", School of Graduate Studies, University of Toronto, 1965.

[19] ERBEN, W.; KEPPLER, J – "A Genetic Algorithm Solving a Weekly Course-Timetabling Problem" – in Proceedings of the 1st International Conference ont the Practice and Theory of Automated Timetabling PATAT ´95, volume 1153 of Lecture Notes in Computer Sciences, 1995

[20] EVEN, S.; ITAI, A.; SHAMIR, A., "On the Complexity of Timetable and Multicommodity Flow Problems", SIAM Journal on Computing, v.5, pp. 691-703, 1976.

[21] FANG, H. – Genetic Algorithms in Timetabling and Scheduling – Ph. D. Thesis, Department of Artificial Intelligence, University of Edinburg, 1994

[22] FELDMAN, R.; GOLUMBIC, M. C., "Interactive Scheduling as a Constraint Satisfiability Problem", Annals of Mathematics and Artificial Intelligence, v. 01, pp. 49-73, 1990.

[23] FERNANDES, C.; Caldeira, J.; Infected Genes Evolutionary Algorithm for School Timetabling. WSES, 2002.

[24] FERRREIRA, A.C.P.L; BRAGA, A. P.; LUDERMIR, T.B. Computação Evolutiva. In: Rezende,S.O.. Sistemas Inteligentes – Fundamentos e Aplicações. São Paulo: Manole, 2003. Cap. 9,p.225-246.

[25] GAREY, M. R.; JOHNSON, D. S.; STOCKMEYER, L., "Some Simplified NP-Complete Problems", in Proceedings of the 6th Annual A.C.M. Symposium on Theory of Computing, pp. 47-63. Também em Theoretical Computer Science, 1, pp. 237-267, 1974.

[26] GOTLIEB, C., "The Construction of Class-Teacher Timetables", in Proceeding of the IFIP Congress, pp. 73-77, 1962.

[27] GROSSO, W., Java RMI. 1 ed. United States of America: O'Reilly, 2001.

[28] GUERVÓS, J.J.M.. Informática Evolutiva: Algoritmos Genéticos.Disponível em < **http://geneura.ugr.es/~jmerelo/ie/ags.htm** >. Acesso em 04/2005.

[29] GUPTA, G.;AKTER, S. F., "Knowledgesheet: A Graphical Spreadsheet Interface for Interactively Developing A Class of Constraint Programs", in

Proc. of PADL 2000, v. 1753 of Lecture Notes in CS, Springer Verlag, pp. 308-323, 2000.

[30] HARALICK, R.; ELLIOT, G., "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", Artificial Intelligence, 14, pp. 263-313, 1980.

[31] HOLLAND, J. "*Adaptation in Natural and Artificial Systems",* The University of Michigan Press, 1th. Edition, 1975.

[32] JUNGINGER, W. "Timetabling in Germany - a survey". Interfaces, v. 16, pp. 66-74, 1986.

[33] KARP, R. M., "Reducibility among Combinatorial Problems", in Complexity of Computer Computations, Miller, R. E., and Thatcher, J. W. (eds), Plenum Press, New York, pp. 85-104, 1972.

[34] LAPORTE, G.; DESROCHES, S., "Examination Timetabling by Computer", Comput. Operations Research, 11, pp. 351-360, 1984.

[35] LIONS, J., "Matrix Reduction Using Hungarian Method for the Generation of School Timetables", Communications of the A.C.M, 9, pp. 319-354, 1966.

[36] LIONS, J., "The Ontario School Scheduling Program", The Computer Journal, 10, pp. 14-21, 1967.

[37] MANVEL, B., "Couloring Large Graphs", in Proceedings of the 1981 Southeast Conference on Graph Theory, Combinatorics and Computer Science, 1981.

[38] METHA, N., "The Application of a Graph Coloring Method to An Examination Scheduling Problem", Interfaces, 11, pp. 57.64, 1981.

[39] MIRANDA,M.N.. Algoritmos Genéticos: Fundamentos e Aplicações.Disponível em < **http://www.gta.ufrj.br/~marcio/genetic.html** >. Acesso em 03/2005.

[40] Object Management Group, Inc. (OMG). Common Object Broker Architecture (CORBA) v3.0. July 2002. Document: formal/04-03-01. Disponível em <http://www.omg.org>. Acesso em 09/2004.

[41] ORFALI, R.; HARKEY, D; EDWARDS, J. Cliente/Servidor Guia Essencial de Sobrevivência. Tradução de Ernesto Lima Veras. 1 ed. Rio de Janeiro: Infobook, 1996.

[42] OSWEGO State University of New York. Biographical Sketches -John Henry Holland. Disponível em <**http://www.cs.oswego.edu/~blue/hx/courses/cogsci1/s2001/section05/subsection5/main.html** >. Acesso em 02/2005.

[43] PACHECO,M.A.C.. Algoritmos Genéticos: Princípios e Aplicações. Disponível em < **http://www.ica.ele.puc-rio.br/publicacoes/download/cnf_0111.pdf** >. Acesso em 03/2005.

[44] POTTS, S.; KOPACK, M. Tradução de Marcos Vieira. Aprenda Em 24 Horas Web Services. 1 ed. Rio de Janeiro : Elseviers , 2003.

[45] RIBEIRO, G.; LORENA, L. A Constructive Evolutionary Approach to School Timetabling. EvoCOP2001 - First European Workshop on Evolutionary Computation in Combinatorial Optimization, 2001.

[46] RICCIOONI, P.R. Introdução a Objetos Distribuídos com CORBA. 1 ed. Florianópolis: Visual Books , 2000.

[47] ROMAN, E.; AMBLER, A.; JEWELL, T. Dominando Enterprise JavaBeans. Tradução de Edson Furmankiewicz. 2 ed. Porto Alegre: Bookman , 2004.

[48] SALVADOR,O.. Introdução a Algoritmos Genéticos.Disponível em < **http://www.freedom.ind.br/otavio/trab/introducao_algoritmos_geneticos.pdf** >. Acesso em 03/2005.

[49] SCHMIDT, G., and STRÖHLEIN, T., “Timetable Construction - An Annotated Bibliography”, Computer Journal, v. 23, N. 04, pp. 307-316, 1980.

[50] School of Earth Sciences, Stanford University. Research - Genetic Algorithm. Disponível em < **http://pangea.stanford.edu/~baris/professional/theoryga.html** >. Acesso em 03/2005.

[51] SILBERSCHATZ, A.; GALVIN, P.; GAGNE, G. Sistema Operacionais Conceitos e Aplicações. Tradução de Adriana Ceschin Rieche. 1 ed. Rio de Janeiro : Campus , 2000.

[52] SILVA,E.E.. Otimização de estruturas de concreto armado utilizando algoritmos genéticos, 2001. Dissertação (Mestrado em Engenharia Civil) – Escola Politécnica-Universidade do Estado de São Paulo. Disponível em < **http://www.teses.usp.br/teses/disponiveis/3/3144/tde-21022002-112505/** >. Acesso em 04/2005.

[53] SOMMERVILLE, I . Engenharia de Software. Tradução de Maurício de Andrade. 6 ed. São Paulo: Addison Wesley , 2003.

[54] Sun Microsystems, Inc. Java IDL Technology Documentation for J2SE 1.4.2. 2003. Disponível em <http://java.sun.com/j2se/1.4.2/docs/guide/idl>. Acesso em 11/2004.

[55] TIMMRECK, E. M., “Scheduler - A Program That Uses Instructor and Student Preferences to Form Timetables”, Computer Science Technical Report #3, University of Wisconsin, 1967.

[56] TIMÓTEO,G.T.S.. Desenvolvimento de um Algoritmo Genético para a Resolução do Timetabling. Disponível em < **http://www.comp.ufla.br/curso/ano2001/Desenvolvimento_de_um_algoritmo_genetico_para_a_resolucao_do_timetabling.pdf** >. Acesso em 05/2005

[57] TRIPATHY, A., “A Lagrangian Relaxation Approach to Course Timetabling”, Journal of the Operational Research Society, 31, pp. 599-603, 1980.

[58] WELSH, D. J. A.; POWELL, M. B., “An Upper Bound for the Chromatic Number of a Graph and Its Application to Timetabling Problems”, Comput. J. 10, pp. 85-86, 1967.

[59] WERRA, D., “Construction of School Timetables by Flow Methods”, INFOR – Canadian Journal of Operations Research and Information Processing, 9, pp. 12-22. 1971.

[60] WERRA, D. de, “An Introduction to Timetabling”, European Journal of Operational Research, 19, pp. 151-162, 1985.

[61] WHITE, G. M., and WONG, S. K. S., “Interactive Timetabling in Universities”, Comput. Educ., v. 12, Nº 4, pp. 521-529, 1988.

[62] YOSHIHAWA, M.; KANEKO, K.; NOMURA, Y. “A constraint-based approach to high school timetabling problems: a case study”. In Proceedings of the 12th National Conference on Artificial Intelligence, pp. 1111-1116, 1994.