UNIVERSIDADE ESTADUAL DO CEARÁ
FRANCISCO WESCLEY CUNHA DE ALMEIDA

UM ALGORITMO GENÉTICO PARA A SOLUÇÃO DE PROBLEMAS ESPECÍFICOS DE PROGRAMAÇÃO INTEIRA

FORTALEZA - CEARÁ
2010

FRANCISCO WESCLEY CUNHA DE ALMEIDA

UM ALGORITMO GENÉTICO PARA A SOLUÇÃO DE PROBLEMAS ESPECÍFICOS DE PROGRAMAÇÃO INTEIRA

Dissertação submetida à Coordenação do Curso de Mestrado Acadêmico em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador:
Prof. Dr. Gerardo Valdísio Rodrigues Viana
Co-orientador:
Prof. Dr. Antônio Clécio Fontelles Thomaz

FORTALEZA - CEARÁ
2010

A447a Almeida, Francisco Wescley Cunha de.

Um algoritmo genético para a solução de problemas específicos de programação inteira / Francisco Wescley Cunha de Almeida. - Fortaleza, 2010.

69p; il.

Orientador: Prof. Dr. Gerardo Valdísio Rodrigues Viana.

Dissertação (Mestrado Acadêmico em Ciência da Computação) – Universidade Estadual do Ceará, Centro de Ciências e Tecnologia.

1. Algoritmo genético. 2. Programação inteira. 3. Otimização combinatória. I. Universidade Estadual do Ceará, Centro de Ciências e Tecnologia.

CDD: 511.8

FRANCISCO WESCLEY CUNHA DE ALMEIDA

UM ALGORITMO GENÉTICO PARA A SOLUÇÃO DE PROBLEMAS ESPECÍFICOS DE PROGRAMAÇÃO INTEIRA

Dissertação submetida à Coordenação do Curso de Mestrado Acadêmico em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Aprovada em 26 / 03 / 2010

BANCA EXAMINADORA

_____

Prof. Dr. Gerardo Valdísio Rodrigues Viana - UECE/UFC
(Orientador)

_____

Prof. Dr. Antônio Clécio Fontelles Thomaz - UECE
(Co-orientador)

_____

Prof. Dr. José Lassance de Castro Silva - UFC
(membro externo)

_____

Prof. Dr. Renato Craveiro de Souza - UFC
(membro externo)

À minha amada esposa, Áfia Suely

DEDICO

AGRADECIMENTOS

A Deus, pela força, coragem e determinação durante o trabalho.

À FUNCAP, pelo apoio financeiro com a manutenção da bolsa de auxílio.

À minha família, pelo incentivo e orações.

À minha esposa, pela compreensão, incentivo e apoio em minha ausência.

Aos colegas da turma de mestrado, pelas sugestões recebidas.

Ao meu co-orientador, Prof. Dr. Antônio Clécio Fontelles Thomaz, inicialmente como orientador, pela sabedoria e confiança em mim depositada no processo de seleção.

Ao meu orientador, Prof. Dr. Gerardo Valdísio Rodrigues Viana, inicialmente como co-orientador, pela paciência e disponibilidade durante a realização deste estudo.

RESUMO

Vários são os algoritmos existentes para solucionar problemas de otimização combinatória. Para modelos que possuam um grande número de variáveis e restrições, e principalmente se essas variáveis são binárias, o tempo de resposta desses métodos se torna impraticável. Diante desta dificuldade e da grande importância prática de tais problemas, vem-se realizando um considerável esforço para se obter melhores soluções. Assim, parte dos trabalhos que enfocam problemas de otimização combinatória utiliza métodos aproximativos, heurísticas e/ou meta-heurísticas. Este trabalho conduz a implementação de um Algoritmo Genético, que é uma meta-heurística populacional cuja eficácia é comprovada pelo grande número de trabalhos publicados na área. O propósito deste trabalho é desenvolver um algoritmo que obtenha boas soluções para algumas instâncias de problemas clássicos de otimização combinatória. Tais problemas – modelados por programação inteira linear – são resolvidos em um tempo computacional aceitável, usando a meta-heurística citada. O desempenho é medido através da qualidade das soluções, do esforço computacional e do tamanho da instância. Possui uma abordagem quantitativa, é, por natureza, uma pesquisa explicativa e faz uso de um método experimental. Utiliza a linguagem de programação Java, em um computador com processador Core 2 Duo e Windows XP como sistema operacional. O trabalho utiliza instâncias extraídas de uma biblioteca de testes largamente utilizada para validação de diversos trabalhos sobre otimização combinatória. O algoritmo proposto obteve boas soluções, próximas da ótima, em um tempo computacional aceitável, devido à qualidade da população inicial e à forma de implementação dos operadores genéticos. Esta última, através de um aumento na taxa de mutação e uma combinação harmoniosa entre técnicas de cruzamento, influenciou de forma significativa nos resultados obtidos.

Palavras-chave: Algoritmo Genético. Programação Inteira. Otimização Combinatória.

ABSTRACT

There are several existing algorithms for solving combinatorial optimization problems. For models that have a large number of variables and constraints, and especially if these variables are binary, the response time of these methods become impractical. Faced with this difficulty and of great practical importance of such problems, has gradually made a considerable effort to obtain better solutions. Thus, several studies that focus on combinatorial optimization problems using approximation methods, heuristics and/or metaheuristics. This work leads the implementation of a Genetic Algorithm, which is a meta-heuristic population whose effectiveness is proven by the large number of published studies. The purpose of this study is to develop an algorithm to obtain good solutions for some instances of classical problems of combinatorial optimization. Such problems, modeled by linear integer programming, are resolved in an acceptable computational time, using meta-heuristics mentioned. Performance is measured by the quality of solutions, the computational effort and the size of the instance. It has a quantitative approach, is, by nature, explanatory research and makes use of an experimental method. It uses the Java programming language, in a computer with Core 2 Duo processor and Windows XP operating system. The work uses instances drawn from a library of widely used tests for validation of several works on combinatorial optimization. The proposed algorithm obtained good solutions, close to the optimum in an acceptable computational time, due to the quality of initial population and how to implement the genetic operators. The latter, through an increase in mutation rate and a harmonious combination of techniques beam, significantly influenced in the results.

Keywords: Genetic Algorithm. Integer Programming. Combinatorial Optimization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A – matriz de entrada binária com m linhas e n colunas;

AG – Algoritmo Genético;

A_j – vetor correspondente à coluna *j* da matriz A;

C_i – configuração de ordem "i";

C – configuração final;

f – Função objetivo;

I – conjunto de todas as linhas (restrições);

J – conjunto de todas as colunas (variáveis);

m – número de restrições;

maxgen – número máximo de gerações;

min () – função que retorna o menor valor de um vetor de números inteiros;

n – Tamanho de cada cromossomo;

npop – Tamanho da população;

$P_{g, i}$ – cromossomo da linha "i" da população de geração "g";

$P_{g, i, j}$ – gene da coluna "j" do cromossomo da linha "i" da população de geração "g";

P_0 – população inicial;

PCC – Problema da Cobertura de Conjuntos;

pcross – Ponto de Cruzamento;

PCV – Problema da Cobertura de Vértices;

PI – Programação Inteira;

PL – Programação Linear;

pm – probabilidade de realizar mutações;

PM – Programação Mista;

px – probabilidade de realizar cruzamentos;

q – probabilidade acumulada;

rnd () – função que gera um número randômico no intervalo [0, 1);

wAG – Algoritmo Genético proposto/desenvolvido;

w_j – peso da coluna *j*;

Σ – Somatório;

$\in$ – Pertence a;

$\lfloor \rfloor$ – função menor inteiro.

SUMÁRIO

1 INTRODUÇÃO

Diariamente, deparamo-nos com problemas de otimização sem nos darmos conta de que sua solução pode ser obtida por um computador facilmente. Poderíamos exemplificar isto quando estamos de partida para uma viagem e precisamos colocar dentro de uma determinada mochila a maior quantidade de itens que nos serão mais úteis, obedecendo sempre sua capacidade máxima. Resolvemos tal problema através de tentativa e erro, colocando e retirando até acharmos uma boa acomodação para a mochila. Tal problema é conhecido no mundo da computação como o problema da Mochila 0/1.

Vários são os algoritmos existentes para solucionar problemas de otimização. *Simplex e Branch and Bound* são exemplos de métodos exatos que encontram soluções ótimas para tais problemas, desde que, em função do tamanho da instância, o tempo seja aceitável. Para modelos que possuam um grande número de variáveis e restrições, e principalmente, se essas variáveis são binárias – problema da Mochila 0/1, por exemplo – o tempo de resposta desses métodos se torna impraticável. Diante desta dificuldade e da grande importância prática de problemas de otimização, vem-se realizando um considerável esforço para se obter melhores soluções. Assim, parte dos trabalhos que enfocam problemas de otimização combinatória utiliza métodos aproximativos, heurísticas e/ou meta-heurísticas. Esta última foi definida por Oliveira (2008) como uma estratégia de alto nível para exploração inteligente do espaço de busca, com o intuito de solucionar, de forma genérica, problemas de otimização. Neste trabalho, conduziu-se a implementação de um Algoritmo Genético (AG), elaborado por Holland (1975) e aprimorada, posteriormente, por Goldberg (1989), que se fundamenta nos conceitos de genética populacional e na teoria da evolução das espécies, conforme Darwin (1859). Um AG é uma meta-heurística populacional que ajuda a solucionar problemas clássicos de otimização, e sua eficácia é comprovada pelo grande número de trabalhos publicados na área. Mais precisamente, um AG é uma técnica de busca que faz uso de uma população inicial de cromossomos, gerada aleatoriamente, que no decorrer das gerações serão selecionados os indivíduos mais aptos para se reproduzirem. Após um número finito de gerações, a população final será formada pelos “melhores” indivíduos onde entre eles está a possível solução para um determinado problema. Desde já, o Algoritmo Genético proposto será denominado de wAG.

Para os experimentos computacionais foram utilizados o Problema da Cobertura de Conjuntos (PCC) pela sua importância comercial na solução de problemas do cotidiano como, por exemplo, o escalonamento de condutores de transportes e o Problema da Cobertura

de Vértices (PCV) pela sua importância no monitoramento de redes, onde a partir dos vértices pertencentes à cobertura, “atinge-se”, ou é possível monitorar, todos os nós da rede (VIANA, 1998, p. 51).

1.1 Objetivos

O propósito deste trabalho é desenvolver um algoritmo que obtenha boas soluções para algumas instâncias de problemas clássicos de otimização combinatória. Tais problemas – modelados por programação inteira linear – são resolvidos em um tempo computacional aceitável, usando a meta-heurística Algoritmo Genético. Esta variedade de aplicações de Programação Inteira é possível, pois, em todos os casos, deseja-se minimizar ou maximizar o valor da função objetivo do problema. O termo “tempo computacional aceitável” está relacionado com o limite de tempo de espera. Para isso, utilizou-se instâncias extraídas da literatura e da biblioteca de testes OR Library proposta por Beasley (1990). Esta biblioteca é largamente utilizada para a validação de diversos trabalhos de otimização combinatória.

Especificamente, acredita-se que uma combinação adequada de algumas técnicas clássicas com outras disponíveis na literatura para o operador de Cruzamento contribuíram para a melhoria dos resultados, assim como uma variação equilibrada na taxa de Mutação. Em particular, esse trabalho serve como um material didático de apoio a leigos no assunto, professores, pesquisadores e estudantes da área. O desempenho do algoritmo é medido através da qualidade das soluções obtidas, do esforço computacional e do tamanho da instância.

1.2 Estrutura do trabalho

O capítulo seguinte apresenta uma revisão da literatura sobre os pontos norteadores deste trabalho: Programação Inteira (PI) e Algoritmo Genético. O capítulo 3 é dedicado à fundamentação teórica onde são expostos os conceitos de Programação Linear (PL); heurísticas e meta-heurística. A forma como o presente trabalho foi conduzido, a metodologia empregada, uma execução detalhada do algoritmo e sua complexidade é mostrada no capítulo 4. Em seguida, são apresentados os resultados computacionais obtidos durante os testes efetuados que fundamentam as conclusões no capítulo 6.

1.3 Artigos publicados

Como forma de divulgação, artigos produzidos a partir deste trabalho foram aceito para publicação nos seguintes eventos científicos:

- XIV Semana Universitária da Universidade Estadual do Ceará.
  período: de 09 a 13 de Novembro de 2009. Fortaleza-Ceará.
- III ERPO NE – III Encontro Regional de Pesquisa Operacional do Nordeste.
  período: de 19 a 20 de Novembro de 2009. Fortaleza-Ceará.
- ALIO/INFORMS – *Association of Latin-Iberoamerican Operational Research Societies / Institute for Operations Research and the Management Sciences, Joint International Meeting,* Buenos Aires, a ser apresentado entre 06 a 09 de junho de 2010.
- XXIV EURO LISBON – XXIV *European Conference on Operational Research*, Lisboa, a ser apresentado entre 11 a 14 de Julho de 2010.

2 REVISÃO DA LITERATURA

2.1 Visão geral

Muitos algoritmos têm sido desenvolvidos para solucionar problemas de Programação Inteira. Para problemas que possuem um pequeno número de variáveis e restrições, a solução pode ser obtida por softwares bastante conhecidos no mercado, tais como LINDO (1994) e LINGO (2008) em suas diversas versões. No entanto, quando o número de variáveis e restrições aumenta, tais *solvers* deixam a desejar no tocante ao tempo computacional, principalmente se as variáveis são binárias. Isso ocorreu na realização dos testes computacionais com o PCC e o PCV, através da Programação Inteira. Como existem muitas aplicações modeladas por estes problemas, diversos algoritmos foram propostos especificamente para solucioná-los.

2.2 Abordagem moderna

De acordo com Oliveira (1999), os algoritmos para a resolução do PCC baseiam-se em Técnicas de Enumeração, Combinações de Técnicas Heurísticas, Relaxações Lineares, Relaxações Lagrangeanas, ou algoritmos baseados em novos paradigmas desenvolvidos de forma abrangente como é o caso das meta-heurísticas. Vários métodos heurísticos foram utilizados para encontrar soluções viáveis para o PCC, no entanto, alguns se mostraram mais eficientes. Dentre eles destacam-se: A heurística gulosa de Chvátal, de Balas e Ho e a de Vasko e Wilson, segundo CHVÁTAL; BALAS & HO; VASKO & WILSON apud Oliveira (1999). Esses métodos são os mais utilizados na construção de soluções viáveis próximas da solução ótima, mas sem garantia de otimalidade.

Nos últimos anos, os AGs têm sido aplicados para um grande número de problemas de Pesquisa Operacional. Chu & Beasley (1996), propuseram um AG específico para o PCC com alterações nos operadores genéticos e a inclusão de um operador heurístico para este tipo de problema. Chu (1997) implementou um AG para solucionar problemas de otimização combinatória. Um ano mais tarde, novamente junto com Beasley, eles usaram um operador especializado para mostrar que o Problema do Particionamento de Conjuntos (PPC), modelado por programação inteira binária, mesmo sendo um problema altamente restrito, onde todas as suas restrições são de igualdade, pode ser solucionado por um AG bem

implementado. Em especial, Levine (1994) aplicou um AG paralelo para solucionar um problema real de *Airline Crew Scheduling* que é um Problema de Particionamento de Conjuntos. Fisher & Kedia (1990) apresentaram um algoritmo de solução ótima baseado em uma heurística dual e aplicaram ao PCC com 2000 variáveis e 200 restrições. Um estudo comparativo de alguns algoritmos aproximativos para o PCC foi conduzido por Grossman & Wool (1995) e Viana (2006). Harche & Thompson (1994) desenvolveram um algoritmo exato, chamado *column subtraction*, que é capaz de resolver instâncias de grande porte do PCC. Viana (1998) propôs um AG para solucionar o problema da Mochila 0/1, a Busca Tabu para o problema do escalonamento de tarefas (*Scheduling*) e a Têmpera Simulada para o problema do empacotamento (*Bin-Packing*). Todos estes procedimentos são meta-heurísticas e os problemas são de PI. Viana et al. (2006) também aborda sobre algoritmos aproximativos para solucionar o PCV e o PCC.

2.3 Considerações

Nepomuceno (2006) utilizou uma combinação de meta-heurísticas para resolver um problema de carregamento de contêiners, Baker (1981) implementou uma combinação de heurísticas para solucionar o problema do PCC na tentativa de melhoria no valor final da solução, assim, os métodos híbridos têm se tornado uma maneira de se obter melhores resultados quando o assunto é encontrar o ótimo para problemas de otimização.

Por tudo isso, percebe-se o vasto campo do conhecimento já percorrido para solucionar problemas de PI através de diversas meta-heurísticas, em especial – Algoritmos Genéticos –, porém, em todas elas, existe a possibilidade de melhoria dos resultados e foi pensando nisto que o presente trabalho foi proposto.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 Conceitos fundamentais

A vida humana é repleta de problemas, porém, estes podem, circunstancialmente, serem resolvidos ou não. Alguns desses são solucionados por meio de um diálogo, outros requerem meios mais eficazes para tanto. Determinados grupos de problemas precisam de uma solução ótima, ou seja, a melhor dentre todas as alternativas possíveis, surgindo, assim, o ato de otimizar que segundo Novo Michaelis (1998), "otimização é o processo pelo qual se determina o valor ótimo de uma grandeza". No entanto, determinados tipos de problemas são humanamente impossíveis de serem resolvidos, daí a necessidade de técnicas mais sofisticadas fizeram dos algoritmos computacionais um aliado eficaz, mas nem sempre eficiente. De acordo com Williams (1999), quando o número de soluções viáveis é muito grande, geralmente astronômico, proveniente de diferentes ordens, o problema de otimização possui uma natureza combinatorial, caracterizando o termo otimização combinatória. Constituem-se em problemas clássicos de otimização combinatória, de acordo com Viana (1998): Programação Linear, Caminho Mínimo numa Rede, Fluxo Máximo numa Rede, Problema de Transporte, Aproveitamento de Corte, Roteamento de Veículos, Carteiro Chinês, Sequenciamento de Máquinas etc.

A modelagem matemática de um problema de otimização é tão importante quanto a sua solução. De um modo geral, um problema do mundo real pode ser modelado de forma a maximizar, ou minimizar, uma função objetivo sujeita a determinadas restrições, expressas por funções matemáticas e relações funcionais. Um problema modelado sob estas condições caracteriza-se como um modelo de Programação Matemática, também conhecido por modelo de Pesquisa Operacional.

Maximizar ou Minimizar ***função objetivo***

Sujeito a ***restrição 1***
restrição 2
...
restrição n

Há vários tipos de modelos de Programação Matemática, dentre eles podemos destacar a Programação Dinâmica, Otimização em Redes, Programação Linear, Programação Não-Linear etc. Cada um tem sua própria forma de ser modelado e o mais comumente utilizado é o de Programação Linear que descreveremos na seção seguinte.

3.2 Programação linear

3.2.1 Definição e modelagem

Para que um problema de Programação Matemática assuma a característica de um problema de Programação Linear (PL), é necessário que a função objetivo e o conjunto finito de restrições sejam representados por funções lineares. Em sua recente obra, Cormen (2002) afirma que para um conjunto de números reais $c_1 , c_2 , \ldots , c_n$ e um conjunto de variáveis $x_1 , x_2 , \ldots , x_n$, uma função linear f sobre essas variáveis é definida por $f(x_1 , \ldots , x_n) = c_1 x_1 , \ldots , c_n x_n$. Qualquer problema de PL pode ser modelado da seguinte forma:

$$\textit{Maximizar ou minimizar} \quad f = \sum_{j=1}^{n} c_j x_j$$

$$\textit{Sujeito a} \quad \sum_{j=1}^{n} a_{ij} x_j = b_i , i = 1 , 2 , \ldots , m$$

$$x_j \geq 0 , j = 1 , 2 , \ldots , n$$

Onde cada x_j, j = 1, 2, …, n, representa uma variável de decisão com valor não negativo; c_j, j = 1, 2, …, n, os coeficientes da função objetivo; $[a_{ij}]$ corresponde à matriz $m \times n$ dos coeficientes tecnológicos, com i = 1, 2, …, m e j = 1, 2, …, n. Quaisquer valores de x_j que satisfaçam todas as restrições constituem uma solução viável. Basta que x_j deixe de satisfazer qualquer uma das restrição para que seja considerado solução inviável. O objetivo é encontrar a melhor solução viável, ou solução ótima, que maximize ou minimize f . (TAHA, 2008, p. 7).

3.2.2 Uma aplicação prática

A Esportes Radicais S/A produz pára-quedas e asas-delta em duas linhas de montagem. A principal linha de montagem tem 100 horas semanais disponíveis para a fabricação dos produtos e a segunda linha tem um limite de 42 horas semanais. Cada um dos produto requer 10 horas de processamento na linha 1, enquanto que na linha 2, o pára-quedas requer 3 horas e a asa-delta requer 7 horas. Sabendo que o mercado está disposto a comprar toda a produção da empresa, bem como que o lucro pela venda de cada pára-quedas é de R$

60,00 e o lucro para cada asa-delta vendida é de R$ 40,00. Qual é a quantidade de pára-quedas e de asas-delta que devem ser produzidos para que a empresa tenha o lucro máximo? (Adaptado de OLIVEIRA, 200-, p. 17).

Verifica-se no problema acima que há duas variáveis de decisão: A quantidade de pára-quedas e a de asas-delta a serem produzidos que serão simbolizados, respectivamente, por X_1 e X_2. A função objetivo a ser maximizada tem a seguinte estrutura: $60X_1 + 40X_2$. O modelo matemático completo deste problema é descrito abaixo:

$$
\begin{array}{ll}
\textit{Max} & 60X_1 + 40X_2 \\
\textit{Sujeito a} & 10X_1 + 10X_2 \leqslant 100 \\
& 3X_1 + 7X_2 \leqslant 42 \\
& X_1 \, , \, X_2 \geqslant 0
\end{array}
$$

Percebe-se do modelo acima que ambas as variáveis de decisão são positivas uma vez que não há quantidades negativas de pára-quedas ou de asas-delta. Também é possível representá-lo graficamente conforme figura abaixo.

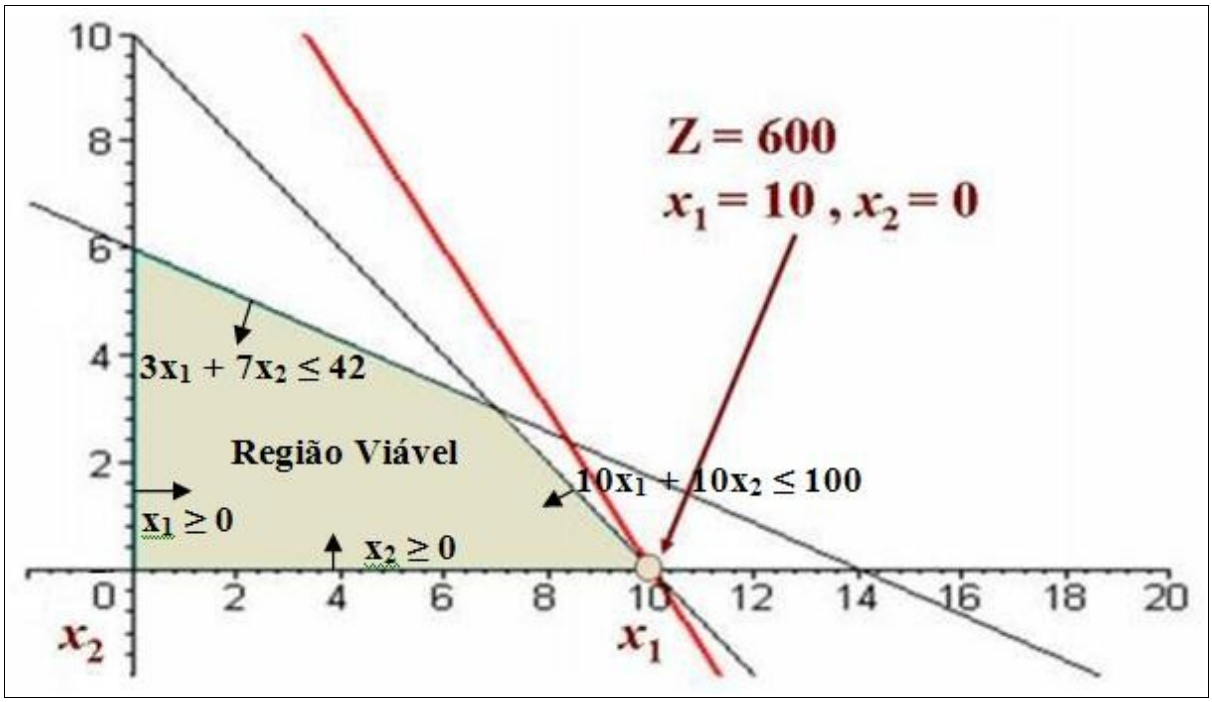

FIGURA 1 – Gráfico de um modelo de Programação Linear (OLIVEIRA, 200-, p. 17)

Assim, o maior lucro que a empresa pode ter é de R$ 600,00, produzindo 10 pára-quedas e nenhuma asa-delta. Esse resultado não a impede de produzir asas-delta, mas, caso isto ocorra, o lucro não será máximo.

3.2.3 Algoritmos exatos de programação linear

Muitos algoritmos foram propostos para dar soluções numéricas de problemas de PL. O mais conhecido, proposto por George Dantzig, é o *Simplex*. Taha (2008) descreve o algoritmo *Simplex* como uma tentativa de passar de um ponto extremo na região de soluções viáveis para um ponto extremo melhor até achar a solução ótima. Essa região de soluções também é conhecida por região factível. Segundo Cormen (2002), o algoritmo se move ao longo do interior da região factível e mantém-se percorrendo os vértices da região viável que é um vértice do *simplex* em cada iteração. Normalmente o algoritmo começa na origem, onde $x_j = 0$, j = 1, 2, …, n. Neste ponto, o valor da função objetivo, f , é zero. Este método aumenta (ou diminui) o valor de $f(x_j)$ que tem a melhor taxa de melhoria em f , um vértice por vez, com o intuito de melhorar a solução.

O primeiro algoritmo de tempo polinomial para solucionar problemas de PL foi o algoritmo de elipsóides. Em contraste com o *Simplex*, temos o algoritmo de pontos interiores, proposto por Karmarkar. A idéia deste é se mover pelo interior da região factível até encontrar a solução final que está em um dos vértices do *simplex*, embora as soluções intermediárias, mesmo sendo viáveis, não sejam vértices do *simplex*.

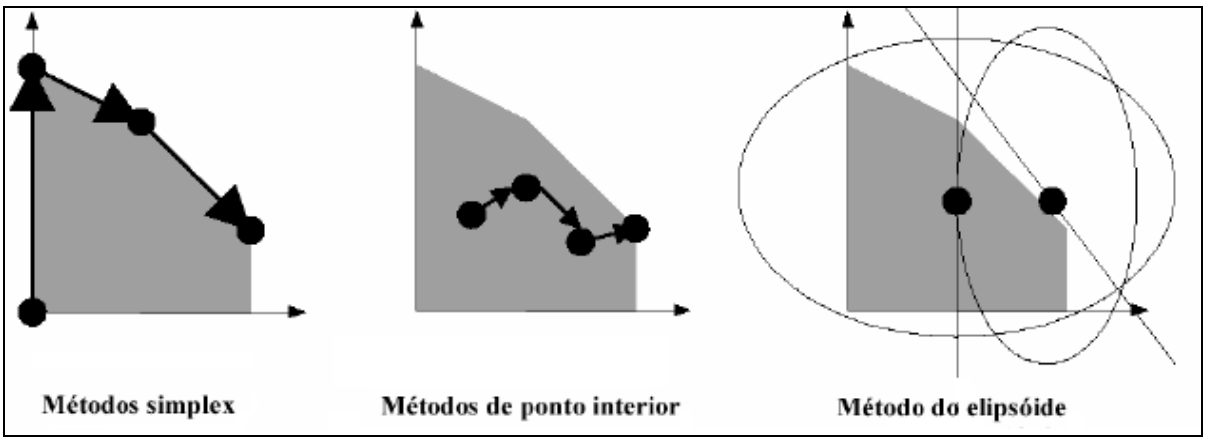

FIGURA 2 – Métodos simplex, de pontos interiores e do elipsóide. (OLIVEIRA, 200-, p. 18)

3.2.4 Programação inteira

A Programação Inteira (PI) é um tipo particular de Programação Matemática onde todas as variáveis estão restritas a valores inteiros, também ditos discretos, conforme Taha (2008). Na Programação Mista, algumas variáveis assumem valores reais e outras, inteiros. A PI ainda pode ser subdividida em duas partes: em uma as variáveis podem assumir quaisquer

valores inteiros e na outra as variáveis assumem apenas valores inteiros binários, ou 0 ou 1. Esse tipo de programação é mais complexo que a PL, no tocante ao elevado tempo computacional, perdendo apenas de problemas de PI 0/1 com restrições de igualdade. Daí a existência na literatura de trabalhos específicos sobre Programação Inteira Binária (ou Programação Inteira 0/1), como é o caso de Chu & Beasley (1996). A Figura 3 apresenta um esquema da Programação Linear e sua classificação.

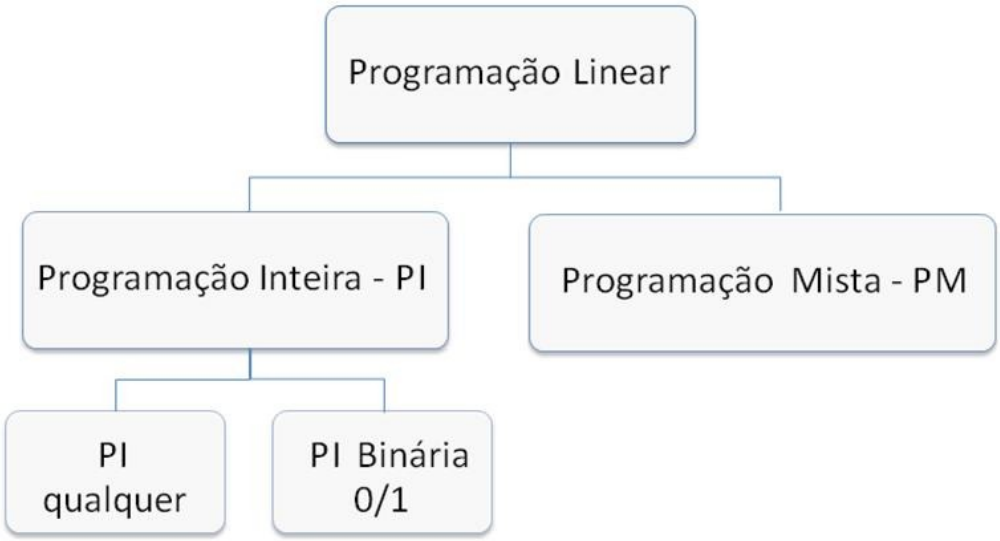

FIGURA 3 – Tipos de Programação Linear

A modelagem matemática de um problema de PI 0/1 é semelhante à de um problema de PL. Segue abaixo um modelo de PI 0/1, sendo que a "igualdade" também pode ser substituída por "desigualdades", dependendo do problema a ser resolvido.

$$\textit{Maximizar ou minimizar } f = \sum_{j=1}^{n} c_j x_j$$

$$\textit{Sujeito a} \quad \sum_{j=1}^{n} a_{ij} x_j = b_i \, , i = 1 \, , 2 \, , \ldots \, , m$$

$$x_j \in \{0 \, , 1\}$$

São problemas típicos de otimização modelados por PI 0/1: Cobertura (Recobrimento) de Conjuntos (Set Covering), Cobertura de Vértices (Vertex Cover), Mochila Binária (Knapsack), Empacotamento (Bin-Packing), Escalonamento de tarefas (Scheduling) etc.

Assim como na PL, existem algoritmos específicos para solucionar problemas de

PI. São eles: o *Branch-and-Bound* e o *Branch-and-Cut*. A estratégia desses algoritmos é descrita por Taha (2008) da seguinte forma: Relaxar a região de soluções da PI, resultando em uma PL normal; Resolver a PL e identificar a solução ótima contínua; A partir desta solução ótima, adicionar restrições especiais que modifiquem iterativamente a região de soluções da PL, resultando em um ponto extremo ótimo que satisfaça as condições de integridade.

3.2.5 O problema da cobertura de conjuntos

O Problema da Cobertura de Conjuntos (PCC), também conhecido por Recobrimento de Conjuntos, é um problema de Programação Inteira 0/1. Seu modelo matemático correspondente é definido da seguinte forma:

$$\textbf{Minimizar } f = \sum_{j=1}^{n} c_j x_j$$

$$\textbf{Sujeito a} \quad \sum_{j=1}^{n} a_{ij} x_j \geqslant 1 \ , i = 1 \ , 2 \ , \ ... \ , m$$

$$x_j \in \{0 \ , 1\}$$

De acordo com Oliveira (1999), $x_j = 1$ se a coluna *j* está na solução e $x_j = 0$, caso contrário. A restrição do modelo indica que cada linha da matriz $[a_{ij}]$ é coberta por pelo menos uma coluna e o fato de x_j assumir apenas valores binários 0/1 indica sua restrição de integridade. Dessa forma, o PCC é o problema equivalente a cobertura das *m* linhas da matriz $[a_{ij}]$, por um subconjunto de suas *n* colunas com custo mínimo, onde c_j é o custo positivo associado a coluna *j*. Se a desigualdade for substituída por igualdade, o problema resultante será chamado de Problema de Particionamento de Conjuntos (PPC). Tal problema é computacionalmente mais difícil de ser resolvido que o PCC.

Vários são os trabalhos encontrados na literatura contendo aplicações do PCC, como: escalonamento de condutores (CASTRO, 2006, p. 18); localização de facilidades (LOPES, 1995, p. 13); seleção de recursos (CORMEN, 2002, p. 815) etc. Para uma maior compreensão, segue abaixo um exemplo, adaptado de Cormen (2002).

Suponha que $X = \{a_1 \ , a_2 \ , \ ... \ , a_{12}\}$ represente um conjunto de 12 habilidades necessárias para resolver um determinado tipo de problema e $Q = \{g_1 \ , g_2 \ , \ ... \ , g_6\}$ um conjunto de 6 pessoas disponíveis para trabalhar nele. Deseja-se formar uma comissão,

contendo o menor número de pessoas possível tal que, para toda habilidade requerida em X, exista um membro da comissão que tenha essa habilidade.

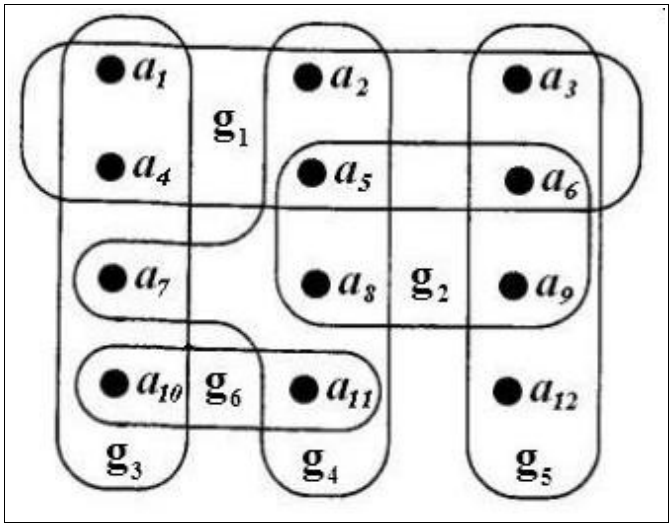

Figura 4: Instância de um Problema de Cobertura de Conjuntos

Pela Figura 4, as habilidades estão representadas pelos pontos pretos e nota-se a quem tais habilidades correspondem: $g_1=\{a_1\ ,\ a_2\ ,\ a_3\ ,\ a_4\ ,\ a_5\ ,\ a_6\}$; $g_2=\{a_5\ ,\ a_6\ ,\ a_8\ ,\ a_9\}$; $g_3=\{a_1\ ,\ a_4\ ,\ a_7\ ,\ a_{10}\}$; $g_4=\{a_2\ ,\ a_5\ ,\ a_7\ ,\ a_8\ ,\ a_{11}\}$; $g_5=\{a_3\ ,\ a_6\ ,\ a_9\ ,\ a_{12}\}$ e $g_6=\{a_{10}\ ,\ a_{11}\}$. Assim, é possível montar a matriz $[a_{ij}]$ dos coeficientes tecnológicos onde cada linha/habilidade é coberta por uma coluna/pessoa:

$$A=\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Como o problema consiste em minimizar o número de pessoas que farão parte da comissão a ser gerada, o modelo matemático do referido problema é esquematizado abaixo:

$$\textit{Minimizar}$$
$$S_1+S_2+S_3+S_4+S_5+S_6$$

$$\textit{Sujeito a}$$
$$S_1+S_3 \geqslant 1$$
$$S_1+S_4 \geqslant 1$$
$$S_1+S_5 \geqslant 1$$
$$S_1+S_3 \geqslant 1$$
$$S_1+S_2+S_4 \geqslant 1$$
$$S_1+S_2+S_5 \geqslant 1$$
$$S_3+S_4 \geqslant 1$$
$$S_2+S_4 \geqslant 1$$
$$S_2+S_5 \geqslant 1$$
$$S_3+S_6 \geqslant 1$$
$$S_4+S_6 \geqslant 1$$
$$S_5 \geqslant 1$$

A solução ótima, ou uma cobertura de conjuntos de tamanho mínimo seria g_3, g_4 e g_5. Pelo menos uma delas está em cada uma das restrições do modelo de PI 0/1 acima. Isto significa que esta comissão, constituída por três pessoas, possuem, juntas, todas as habilidades de X, portanto, são suficientes para resolver o problema.

3.2.6 O problema da cobertura de vértices

Seja G = (V, E) um grafo não orientado com vértices em V. Uma Cobertura de Vértices é um subconjunto $V' \subseteq V$ tal que, se (u, v) é uma aresta de G, então $u \in V'$ ou $v \in V'$ (ou ambos), (CORMEN, 2002, p. 808). O referido autor afirma ainda que o tamanho de uma Cobertura de Vértices corresponde ao número de vértices que ela contém. Ratificando tal definição, Viana (1998) afirma que V' é uma Cobertura de Vértices se toda aresta de G possuir pelo menos uma de suas extremidades em V'. Seu modelo de Programação Matemática correspondente pode ser descrito por:

$$\textit{Minimizar } f = \sum_{i=1}^{n} c_i x_i$$
$$\textit{Sujeito a} \quad x_i + x_j \geqslant 1 \quad (i\ ,\ j) \in E$$
$$x_i \in \{0\ ,\ 1\} \quad i \in V$$

De modo análogo ao PCC, $x_i = 1$ ou $x_j = 1$ (ou ambos) se a coluna *i* ou *j* (ou ambas) está na solução. $x_i = 0$ e $x_j = 0$ indicam que a aresta *(i, j)* não faz parte da solução. A restrição do modelo indica que cada linha da matriz $A_{m \times 2}$, sendo *m* o número de arestas do grafo G, é coberta por pelo menos uma coluna e o fato de x_i assumir apenas valores binários 0/1 indica sua restrição de integridade.

Diversos trabalhos podem ser encontrados na literatura contendo aplicações do PCV. Como exemplo, tem-se o problema do monitoramento de redes (VIANA, 1998, p. 51), onde a partir dos vértices pertencentes à cobertura "atinge-se", ou é possível atingir, monitorar todos os nós da rede. Para uma maior compreensão, segue abaixo um exemplo, adaptado de Cormen (2002).

Suponha que na Figura 5 estejam representados 7 computadores interligados por uma rede. Os nós dessa rede, que são os computadores, correspondem aos vértices de um grafo G = (V, E), onde $V = \{a, b, c, d, e, f, g\}$. O problema consiste em determinar quais vértices/computadores serão escolhidos para a função de servidor, de modo que cada computador deva estar, obrigatoriamente, ligado a pelo menos um servidor.

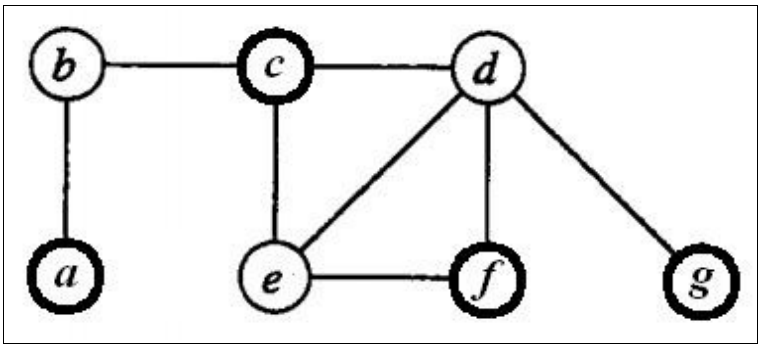

Figura 5: Instância de um Problema de Cobertura de Vértices

Assim, é possível montar uma matriz $[a_{ij}]$ de coeficientes tecnológicos onde cada linha/aresta é coberta por pelo menos uma coluna/vértices da seguinte forma:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Como o problema consiste em minimizar o número de vértices que farão parte da cobertura de tamanho mínimo, o modelo matemático do referido problema é esquematizado da seguinte forma:

Minimizar
$$a + b + c + d + e + f + g$$

Sujeito a
$$a + b \geqslant 1$$
$$b + c \geqslant 1$$
$$c + d \geqslant 1$$
$$c + e \geqslant 1$$
$$d + e \geqslant 1$$
$$d + f \geqslant 1$$
$$e + f \geqslant 1$$
$$d + g \geqslant 1$$

Para uma melhor compreensão, a rotulação dos vértices foi mantida. A cobertura de vértices ótima para esse problema contém apenas três vértices: *b*, *d* e *e*, ou seja, $V' = \{b\ ,\ d\ ,\ e\}$. Isto significa que os três vértices são suficientes para resolver o problema, pois todos os computadores desta rede estão ligados a pelo menos um deles.

3.3 Heurísticas

3.3.1 Definição e aplicações

O uso de instâncias cada vez maiores pode ocasionar uma demora no tempo de execução de algoritmos ou, sobretudo, a impossibilidade de serem executados devido à complexidade. Sendo assim, faz-se necessário o uso de métodos específicos para resolvê-los. Heurística nada mais é do que qualquer método ou técnica criada, ou desenvolvida, para resolver um determinado tipo de problema, (VIANA, 1998, p. 91). Podem ser aplicadas tanto a problemas de PL, quanto a PI ou a PM, tais como: Roteamento de Veículos, Caixeiro Viajante; Particionamento de Clique, Particionamento e Coloração de Grafos; Alocação de Salas, Localização; Mochila Binária, Programação Não-Linear; Cobertura e Particionamento de Conjuntos; Telecomunicações, Tecnologia; Lógica e Inteligência Artificial; Produção, Estoque e Investimentos dentre outras.

Taha (2008) apresentou dois algoritmos heurísticos para solucionar problemas de PI, são eles: o do vizinho mais próximo e o do subcircuito inverso. O segundo é mais eficiente, embora requeira mais cálculos. As duas heurísticas trabalham de forma combinada, uma vez que a primeira produz como saída a entrada para a segunda. Oliveira (1999) apresenta várias heurísticas que solucionam o Problema da Cobertura de Conjuntos, fazendo uma comparação numérica entre elas através das heurísticas de Chvàtal (1979), de Balas & Ho (1980) e de Vasko & Wilson (1984), dentre outras.

3.3.2 Classificação

De acordo com Viana (1998), as heurísticas podem ser assim classificadas:

- *Gulosas ou Míopes*: Baseiam-se no incremento da solução a cada passo, em que o elemento a incluir é determinado por uma função usada para guiar a busca denominada função heurística. Míope no sentido de não ter inteligência suficiente para escapar por si só de ótimos locais. Tem a vantagem de ser extremamente rápida e produzir soluções razoavelmente eficientes;
- *Locais*: Focam a obtenção da solução, não importando o caminho percorrido dentro do espaço de soluções. O mais importante é o que está ao seu redor, ou seja, na sua vizinhança. Têm a mesma desvantagem das gulosas, ficando presas a ótimos locais;
- *Partição ou Grupamento*: Segundo Branco (1984), o espaço de soluções é dividido em *n* subconjuntos (ou subespaços) e uma heurística qualquer percorrerá cada um destes *n* subconjuntos para ao final escolher-se a melhor solução dentre os subconjuntos que foram particionados.

Há ainda outros tipos de heurísticas como: Híbridas, que resultam da combinação de duas ou mais heurísticas, segundo Alvin (2003); de Construção, que possuem a característica de construir uma solução inicial e as de Melhoria, que partem de uma solução já existente com o objetivo de aperfeiçoá-la.

3.3.3 Heurísticas utilizadas neste trabalho

Foram utilizadas duas heurísticas para gerarem a população inicial do wAG para o PCC e o PCV: uma aleatória, que é bastante simples, e outra gulosa. Tais procedimentos serão detalhados na seção 3.5.2.

3.4 Meta-heurísticas

Meta-heurísticas são heurísticas de uso geral, ou ainda, são métodos heurísticos para resolver, de forma genérica, problemas de otimização. Viana (1998) afirma que uma meta-heurística explora o espaço de configurações – espaço de todas as possíveis soluções – alternando entre os procedimentos de Intensificação e Diversificação a fim de evitar uma parada prematura em ótimos locais (máximo/mínimo local), proporcionando melhores soluções. Ainda segundo o mesmo autor, toda meta-heurística faz uso de uma heurística – daí ser conhecida também como heurística das heurísticas – para realizar o processo de Intensificação e de técnicas probabilísticas para o de Diversificação, apresentando, assim, soluções bem próximas da ótima.

Uma boa meta-heurística deve possuir as seguintes características: Adaptação a instâncias especiais; escapar de ótimos locais; fazer uso da estrutura do problema; utilizar uma estrutura de dados adequada; possuir boas técnicas para construção das soluções iniciais viáveis; reinicializar procedimentos de forma dinâmica; melhoria de solução através de busca local; diversificação de busca quando nenhuma melhoria adicional for possível; intensificação da busca em regiões promissoras; randomização controlada na alternância entre a Intensificação e a Diversificação; se necessário, realizar um pré-processamento.

São exemplos de meta-heurísticas: *Tabu Search* (Busca Tabu) (VIANA, 1998, p. 111), *Simulated Annealing* (Têmpera Simulada) (VIANA, 1998, p. 97), Algoritmo Genético (VIANA, 1998, p. 127), GRASP (*Greedy Randomized Adaptative Search Procedure*), *Ant System* (Colônia de Formigas), *Scatter Search*, *Path-Relinking*, *MultiStar*, *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND), *Variable Neighborhood Search* (VNS) etc. Este trabalho abordará apenas a meta-heurística Algoritmo Genético.

3.5 Algoritmo genético

3.5.1 Conceitos básicos

Um Algoritmo Genético (AG) baseia-se nos processos observados na evolução natural das espécies. A terminologia dos sistemas biológicos tem correspondentes no modelo computacional como: cromossomos, genes, alelos, genótipos e fenótipos. A partir de uma determinada população, os melhores indivíduos sobrevivem e geram descendentes com suas características hereditárias, de maneira similar à teoria biológica dos sistemas naturais. Da mesma forma, um AG parte de uma população de indivíduos gerados aleatoriamente, faz a avaliação de cada um, seleciona os melhores e promove manipulações genéticas, como Cruzamento (Diversificação) e Mutação (Intensificação) a fim de criar uma nova população. Vale ressaltar que o termo "melhores" representa aqueles cromossomos cuja função de custo tenha os maiores (maximização) ou menores (minimização) valores, de acordo com Viana (1998). Este processo adaptativo pode ser usado para resolver alguns problemas de otimização, inclusive problemas de Programação Inteira.

Segundo Holland (1975) apud Viana (1998), os Algoritmos Genéticos possuem características que consolidam seu uso, tais como: uso de técnicas de randomização, robustez, foco no valor da função objetivo – o que permite experimentos com modelos não-lineares –, uso exclusivo de regras probabilísticas e, conforme citado na seção anterior, são meta-heurísticas, ou seja, heurísticas de uso geral. Em sua dissertação de mestrado, Viana (1998) descreve os passos necessários para a construção de um AG:

1. Escolher a forma de representação dos cromossomos. Geralmente, utilizam-se strings ou vetores, neste trabalho, utilizou-se um vetor de inteiros;
2. Geração da população inicial. Esta geração ocorre aleatoriamente, uma vez que essa é uma das características de um Algoritmo Genético. O tamanho desta população, bem como das outras que serão geradas, é de fundamental importância, visto que, se tal parâmetro é pequeno o algoritmo converge rapidamente e geralmente não se obtém bons resultados e se for grande, pode comprometer o tempo de execução do algoritmo, como aconteceu com algumas das instâncias utilizadas para testes neste trabalho; A heurística utilizada para a geração da população inicial está descrita na seção 3.6.2;
3. Avaliação da função objetivo. Esta avaliação é feita para cada indivíduo da população e ela é de fundamental importância para se verificar a viabilidade da solução, no

entanto, há um consumo de tempo considerável quando da execução do algoritmo;

4. Utilização de operadores genéticos tais como: Reprodução, Cruzamento e Mutação.

De acordo com Chu & Beasley (1996), um AG é descrito da seguinte forma:

Geração da população inicial;
Avaliação da aptidão dos indivíduos na população;
Repita
Seleção dos melhores indivíduos;
Cruzamento;
Mutação;
Avaliação da aptidão dos indivíduos gerados;
Substituição de alguns ou todos os indivíduos;
Até uma solução satisfatória ser encontrada.

Após a população inicial ter sido gerada e avaliada pela função objetivo, alguns indivíduos - os mais aptos - serão selecionados e cruzados. Logo em seguida, uma pequena minoria passará por uma mutação e finalmente os "sobreviventes" serão escolhidos para fazer parte de um grupo chamado "possível solução". Possível porque mediante a etapa de Reprodução tal indivíduo pode ser substituído pelo surgimento de um melhor do que ele. Novamente eles serão avaliados e os mais fracos darão lugar aos mais fortes. Fracos e fortes dizem respeito a sua aptidão ou valor da função objetivo.

3.5.2 Geração da população inicial

De acordo com Oliveira (1999), O primeiro passo para implementar com sucesso um Algoritmo Genético para um problema particular é conceber um esquema de representação e usá-lo para um mapeamento entre o espaço de soluções e o espaço de estruturas, de modo a preservar o significado do problema original. Para uma melhor compreensão, será considerado aqui o exemplo aplicado da seção 3.2.5.

A representação binária 0/1 é uma escolha usual e óbvia para o PCC por representar as variáveis básicas inteiras 0/1. Usa-se um vetor de *n* dígitos binários como estrutura de cromossomo onde *n* é o número de colunas no PCC. Conforme mencionado anteriormente, o valor de 1 para dígito implica que a coluna *j* está na solução. A representação

binária de um cromossomo/comissão para o PCC da seção 3.2.5 é ilustrado na Figura abaixo:

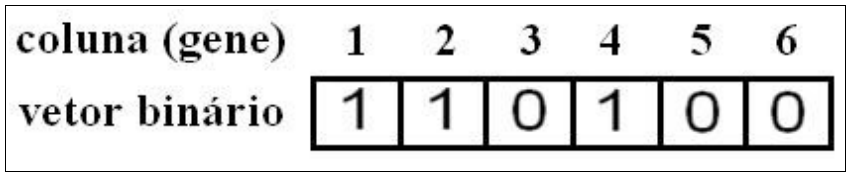

FIGURA 6 – Representação binária de um cromossomo

Neste caso, a Figura 6 indica que g_1 , g_2 e g_4 foram escolhidos para fazer parte da comissão, no entanto, este cromossomo não é viável uma vez que nem todas as habilidades necessárias para realizar a tarefa são satisfeitas, nenhuma das pessoas desta comissão possui as habilidades a_{10} e a_{12} .

A aptidão de um cromossomo está diretamente relacionado com o valor numérico de sua função objetivo. Com a representação binária, a aptidão $f(C_i)$ de um indivíduo *i* é calculado simplesmente por:

$$f(C_i) = \sum_{i=1}^{n} w_i C_i$$

onde C_i é o valor da *i-ésima* coluna no vetor binário e w_i é o custo da coluna *i*. Assim, a aptidão do cromossomo da Figura 6 é 3.

Como a população é constituída por vários cromossomos, o wAG gerou a população inicial de duas formas distintas. Em primeiro lugar, conforme mencionado na seção 3.4.3, um procedimento simples monta, aleatoriamente, a estrutura cromossômica com 0 ou 1, elemento a elemento. Tal procedimento baseia-se na heurística utilizada por Chu & Beasley (1996), com algumas modificações. A maior parte da população é gerada por ela da seguinte forma:

Considere

w_j = peso da coluna *j*;

A = matriz de entrada binária com *m* linhas e *n* colunas;

A_j = vetor correspondente à coluna *j* da matriz A;

I = conjunto de todas as linhas (restrições);

J= conjunto de todas as colunas (variáveis);

S = conjunto de colunas em uma solução;

C = cromossomo formado por S.

Heurística Aleatória:
Inicialize $S = \emptyset$ e $C_j \leftarrow 0, \forall j \in J$,
Para cada linha $i \in I$ **faça**
Selecione aleatoriamente uma coluna $j \in J$,
Se $j \notin S$, **então**
$S \leftarrow S \cup \{j\}$,
$C_j \leftarrow 1$,
fim-se
fim

O procedimento acima é repetido *npop* vezes, ou seja, a quantidade máxima da população inicial. A heurística inicializa afirmando que o conjunto de colunas em uma dada solução é vazio e o cromossomo C, formado por este conjunto S, possui todos os seus genes iguais a zero. Vale ressaltar que esta heurística é específica para um problema de Programação Inteira binária. O primeiro laço seleciona aleatoriamente uma coluna j pertencente ao conjunto de todas as variáveis do problema e adiciona ao conjunto de colunas formadoras da solução, se este conjunto S já não contiver aquela coluna j e isto para cada uma das restrições. Em seguida, cada gene do cromossomo C cujas posições correspondem aos elementos formadores de S é substituído pelo valor um. Esse procedimento evita o uso de uma segunda heurística para testar a viabilidade do cromossomo uma vez que este foi construído a partir de cada linha do conjunto, atendendo às restrições e, portanto, viável.

Com o objetivo de ter uma população inicial bem diversificada, dois elementos da população são gerados pela segunda heurística. O procedimento abaixo é repetido duas vezes:

Heurística Gulosa:
Inicialize $C_j = 0, \forall j \in J$,
Enquanto $I \neq \emptyset$ **faça**
$$j \leftarrow \ argmin\left\{ k\,,\ \frac{w_k}{|A_k|},\ |A_k| \neq 0 \right\}$$
$C_j \leftarrow 1$,
Para cada linha $i \in I$ **faça**
Se $a_{ij} = 1$ **então**
$I \leftarrow I - \{i\}$,
fim-Enquanto
fim

Estes dois elementos que são gerados pela Heurística Gulosa diferenciam-se pelo peso: em uma execução é fixado o valor 1 para cada w_k, k = 1,..., n; na outra é fixado o peso real de cada coluna. Para instâncias sem peso (ou unicusto) serão gerados dois elementos

iguais. Pela característica dos AGs, esta peculiaridade na população inicial não trará problemas na sua evolução.

A heurística inicializa afirmando que o cromossomo C, formado pelo conjunto de colunas na solução, possui todos os seus genes iguais a zero. De um modo geral , o objetivo do laço externo é decrementar as linhas à medida que determinada coluna das mesmas são selecionadas para fazer parte da solução. Isto acontece da seguinte forma: a função argumento mínimo (argmin) retorna o índice *k* que atinge o valor mínimo referente à razão entre o peso e o valor absoluto da matriz de entradas binárias, da coluna *k*. Este índice, que agora faz parte do conjunto S, corresponde ao gene do cromossomo C que recebe o valor 1. Em seguida, o laço interno concretiza a eliminação de todas as linhas "cobertas" pela coluna selecionada.

3.5.3 Reprodução

Só faz sentido reproduzir os melhores indivíduos selecionados antes do término do algoritmo. Com base nos valores da função objetivo, os melhores cromossomos, considerados mais fortes, são selecionados e duplicados em substituição aos piores, considerados mais fracos. Para isto, este trabalho utiliza uma "roleta viciada", de forma que cada cromossomo seja copiado um número de vezes proporcional ao valor da função objetivo, de acordo com Viana (1998), ou seja, consiste em realizar a seleção dos cromossomos semelhante ao que ocorre nos jogos de azar. O método da roleta consiste em associar a cada indivíduo uma fatia da roleta igual à sua probabilidade de sobrevivência ou seleção. Existem ainda outros métodos de reprodução como: método da roleta múltipla ou amostragem universal estocástica, que consiste numa variação da roleta viciada em que *n* elementos são selecionados ao mesmo tempo (LUDESCHER, 2008, p. 6) e o método do torneio que:

> Consiste em selecionar n indivíduos da população aleatoriamente com a mesma probabilidade. Entre estes n cromossomos aquele com maior aptidão são selecionados para a população intermediária. O processo se repete até a população intermediária ser preenchida (LUDESCHER, 2008, p. 6).

Considerando o mesmo exemplo aplicado da seção 3.2.5 para o PCC, a tabela seguinte representa uma população constituída por cinco cromossomos binários com suas respectivas aptidões e, na figura, a sua respectiva roleta viciada.

TABELA 1 - Indivíduos de uma população para um problema de minimização

i	**Configuração** C_i	**Aptidão** $\frac{1}{f(C_i)}$	**Aptidão Relativa** p_i	**Aptidão Acumulada** q_i
1	101101	0,25	0,21	0,21
2	111111	0,16	0,13	0,34
3	100110	0,33	0,28	0,62
4	101111	0,20	0,17	0,79
5	110101	0,25	0,21	1,00

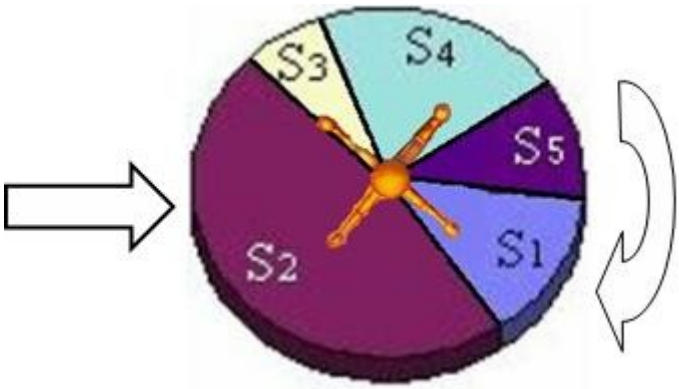

FIGURA 7 – Roleta viciada para a fase de Reprodução

A aptidão relativa, também conhecida por probabilidade de seleção p_i é calculada da seguinte forma:

$$p_i = \frac{f(C_i)}{\sum_{i=1}^{npop} f(C_i)}$$

O limite superior do somatório, *npop* , para este exemplo é igual a 5. Viana (1998) simula o funcionamento da roleta, que se assemelha a um gráfico de pizza muito utilizado em Estatística, gerando um número aleatório "r" no intervalo de 0 a 1 e comparando seu valor com a probabilidade acumulada q_i, também chamada de aptidão acumulada, considerando $q_0 = 0$. Assim, se $q_{i-1} < r \leqslant q_i$, deve-se selecionar o indivíduo S_i.

$$q_j = \sum_{i=1}^{j} p_i$$

Como o PCC é um problema é de minimização, considera-se a aptidão como sendo $\frac{1}{f(C_i)}$, no entanto, se tivéssemos um problema de maximização, a aptidão seria apenas $f(C_i)$. Maiores detalhes sobre o funcionamento da reprodução do wAG pode ser encontrado na seção 4.2.

3.5.4 Cruzamento

O Cruzamento (*crossover*) é o operador que possibilita a recombinação das estruturas genéticas da população, com o objetivo de gerar cromossomos diferentes dos atuais. Com isso há uma diversificação do espaço de configurações, permitindo regiões ainda não exploradas. O cruzamento ocorre com a troca de parte do padrão genético de dois cromossomos. Em termos computacionais, há a troca das "caudas" de dois vetores, gerando, assim, dois vetores diferentes dos originais. O ponto que antecede a "cauda" é conhecido por ponto de quebra ou ponto de corte (*pcross*). Também podem ser usados dois ou mais pontos de quebra com o intuito de melhorar a diversificação.

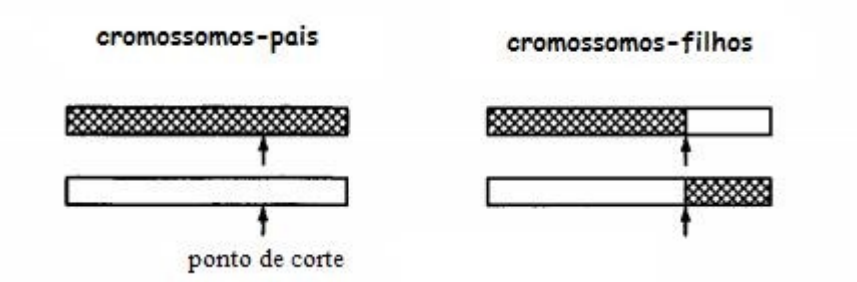

FIGURA 8 – Operador de Cruzamento básico (CHONG, 2001, p. 240)

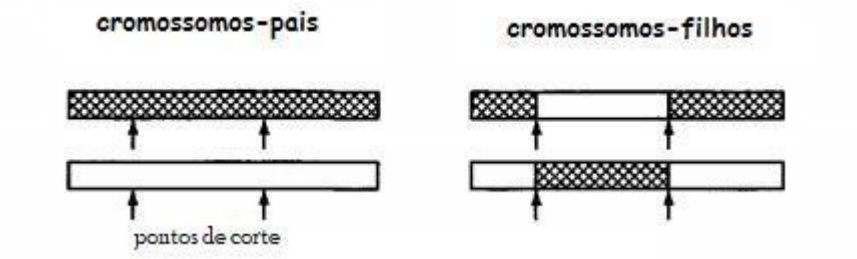

FIGURA 9 – Operador Cruzamento com dois pontos de corte (CHONG, 2001, p. 240)

Considerando o exemplo da seção 3.2.5, referente ao PCC, a troca do padrão genético de dois cromossomos será a troca das habilidades que cada uma das duas pessoas

possui. Assim, os indivíduos gerados possuirão uma "junção" de habilidades de seus genitores.

Segundo Viana (1998), o cruzamento pode ser realizado com dois a dois elementos da população inteira, contudo, é mais comum atribuir um percentual na faixa de 25% a 75% da população. Os dois casos foram testados e o processo de convergência é mais lento quando do uso da população inteira. Melhores resultados foram obtidos aumentando-se o percentual do operador Mutação.

Chu & Beasley (1996) utilizam um operador chamado *fusion* (fusão) em substituição ao operador *crossover* clássico. O operador *fusion* não possui pontos de quebra, ele apenas faz uma fusão entre dois cromossomos-pais, C_1 e C_2, gerando um único cromossomo-filho, C. Este trará sobre si a carga genética dos dois cromossomos-pai baseados na estrutura e na aptidão de cada um. Possui melhor aptidão aquele cromossomo com a maior probabilidade de ceder seu gene, e esta é feita com base no melhor valor da função objetivo.

Considerando $f(C_1)$ e $f(C_2)$ a aptidão dos cromossomos-pais e um dado problema de minimização, então para todo j = 1, …, n, onde n é o tamanho de cada cromossomo-pai:

Se $C_{1,j} = C_{2,j}$ **então** $C_j \leftarrow C_{1,j} = C_{2,j}$ **fim**
Se $C_{1,j} \neq C_{2,j}$ **então**
$C_j \leftarrow C_{1,j}$ com probabilidade $p = \dfrac{f(C_2)}{f(C_1) + f(C_2)}$,
$C_j \leftarrow C_{2,j}$ com probabilidade $1 - p$
fim

Este trabalho conduziu a implementação do cruzamento através de dois pontos de quebra juntamente com o operador *fusion*, 50% para cada técnica. Verificou-se, com base nos testes, que esta forma híbrida possui melhor desempenho do que a escolha de apenas uma.

3.5.5 Mutação

A Mutação corresponde a uma pequena perturbação na estrutura do cromossomo com o intuito de regenerar algum indivíduo que foi eliminado de forma inesperada. No caso do exemplo da seção 3.2.5 para o PCC, a perturbação na comissão pode regenerar alguma pessoa que foi eliminada de forma precoce. Tanto o percentual de mutação quanto a escolha do gene de um determinado cromossomo a ser alterado pode ser feita segundo sugeriu Viana (1998), cerca de 1% de todos os genes, embora melhores resultados são obtidos quando esta

taxa se eleva até 10%. Chu & Beasley (1996) utilizam uma taxa de mutação variável, pois acreditam que, à medida que o AG converge, os cromossomos ficam parecidos uns com os outros, necessitando, assim, de uma taxa maior de mutação. Este trabalho também conduziu a implementação de uma taxa de mutação variável ou adaptativa, no decorrer das gerações, mas não igual à proposta por Chu & Beasley (1996) e sim, semelhante à proposta de Lopes (1995), calculado pela expressão:

$$p_m = p_f * \exp\left(\frac{\eta * (t - maxgen)}{maxgen}\right)$$

Onde p_f é um valor pré-estabelecido para a taxa de mutação final, que aqui será de 10%, η é uma constante cujo valor é igual a 2,3025 escolhida de forma a inicializar a taxa de mutação em 1%, t é a geração corrente e *maxgen* é o número máximo de gerações. Assim, o algoritmo inicia com uma taxa de mutação pequena e varia exponencialmente ao longo das gerações. O valor de 1% para a taxa inicial de mutação foi escolhido com base em Viana (1998), já o valor final de 10% foi escolhido com base na observação dos testes computacionais.

De um modo geral, a mutação ocorre pela troca de um gene por outro. No caso do exemplo da seção 3.2.5 para o PCC, a troca é de uma pessoa por outra. Ela também pode ocorrer de outras formas, caso em que o gene é um número real. Hong (2000), sugere quatro formas de mutação, uma delas é escolher aleatoriamente um gene e trocá-lo, se estiver representado pelo *bit* 1, troca-se pelo *bit* 0 e vice-versa, no caso de o cromossomo ser uma cadeia de *bits* 0/1. Outra forma seria inverter todos os *bits*, a diferença entre ambas é que, na primeira apenas um *bit* é trocado, na segunda, todos os *bits* são trocados. As outras duas formas de mutação ocorrem quando cada gene do cromossomo é representado por números reais, que está fora do escopo deste trabalho.

1ª forma	**2ª forma**
Antes da Mutação: 0 0 1 1 **0** 0	Antes da Mutação: **0 1 1 0 0 1**
Depois da Mutação: 0 0 1 1 **1** 0	Depois da Mutação: **1 0 0 1 1 1**

Percebe-se, acima, que na 1ª forma a mutação ocorreu na 5ª posição do cromossomo, ou seja, o C_5 foi regenerado e sua entrada na comissão possibilitou o encontro da solução ótima para o problema. Já na 2ª forma, em todas as posições. Este trabalho utilizou

a primeira forma de mutação, diferente do que ocorre na segunda forma, uma vez que a troca de todos os bits ocasiona uma diversificação do espaço de buscas, trabalho este já realizado pelo operador Cruzamento.

Com base nos experimentos, para problemas de minimização, verificou-se que a substituição do bit 1 pelo bit 0 e a não substituição do bit 0 pelo bit 1, acarreta uma melhora significativa no resultado final. Mas isso, para uma pequena taxa de mutação. Assim, tal procedimento foi implementado nos primeiros 90% das gerações. O mesmo vale para problemas de maximização, alterando a sequência de substituições acima.

4 ALGORITMO PROPOSTO

4.1 Material e métodos

De acordo com Silva & Menezes (2001), este trabalho possui uma abordagem quantitativa, uma vez que todos os resultados podem ser traduzidos em números e informações para classificação e análise, sem desmerecer a qualidade de tais resultados e fazendo uso de algumas técnicas estatísticas (percentagem e média aritmética). Além disso, do ponto de vista de seus objetivos, este trabalho trata-se de uma pesquisa explicativa, pois identificam os fatores que contribuem para a eficiência do AG na solução de problemas de Programação Inteira. Faz uso de um método experimental cujo objeto de estudo é o AG, onde as variáveis capazes de influenciá-lo são a população inicial e os operadores genéticos. As formas de controle e de observação dos efeitos que as variáveis produzem no objeto são bem definidas nos resultados computacionais obtidos.

Foi utilizada a linguagem de programação Java, em um notebook com processador Core 2 Duo e Windows XP como sistema operacional. Para os testes computacionais, o software LINDO (1994) Industrial e LINGO (2008) foram solicitados para testar algumas instâncias. Algumas delas foram extraídas de livros da área estudada e da Biblioteca de Pesquisa Operacional OR Library (BEASLEY, 1990), uma biblioteca de testes largamente utilizada para validação de diversos trabalhos sobre otimização combinatória.

4.2 Algoritmo genético proposto - wAG

O wAG seguiu os passos sugeridos em Viana (1998, p.143 e 144, com adaptações). Considerando-se os seguintes indicadores:

P_0: população inicial representada por uma matriz com *npop* linhas (cromossomos) por *n* colunas (número de genes em cada linha);
$P_{g,i}$: cromossomo da linha “i” da população de geração “g”;
$P_{g,i,j}$: gene da coluna “j” do cromossomo da linha “i” da população de geração “g”;
C_i: configuração de ordem “i”;
C: configuração final;
i, j, g: variáveis de controle de laços;

K, L, R: variáveis auxiliares;

rnd (): função que gera um número randômico no intervalo [0, 1);

min (): função que retorna o menor valor de um vetor de números inteiros;

⌊ ⌋: função menor inteiro;

q: probabilidade acumulada.

```
inicio do algoritmo
    inicio do método principal
        leitura ("instancia.txt")                          {arquivo externo}
        parametros (npop, n, maxgen, px, pm)
        geração(P0)                                        {população inicial}
        ngens = npop * n
        g = 0
        repita
            g = g + 1
            Pg = reprodução (Pg-1)
            i = 0
            repita
                i = i + 2
                se (rnd() ≤ px) então
                    cruzamento (Pg, i-1, Pg, i)
                fim-se
            até (i ≥ npop)
            i = 0
            repita
                i = i + 1
                se (rnd() ≤ pm) então
                    L = ⌊i/n⌋
                    K = i – L * n
                    Pg, L = mutação (K, Pg, L)
                fim-se
            até (i ≥ ngens)
            C = seleção(Pg)
        até (g ≥ maxgen)                                   {condição de parada}
        imprima (C)                                        {melhor solução}
    fim do método principal

    inicio do método geração(P0)                           {gera a população inicial}
        P0, 1 = heurística gulosa com o peso igual a 1 da seção 3.4.3
        P0, 2 = heurística gulosa com o peso real da seção 3.4.3
        i = 3
        repita
            P0, i = heurística aleatória da seção 3.4.3
            i = i + 1
        até (i ≥ npop)
    fim do método geração
```

inicio do método reprodução (P_{g-1}) {(roleta viciada) seleciona os indivíduos mais aptos da população}
i = 0
repita
R_i = rnd()
se ($0 < R_i \leq q_0$) **então**
$P_{g-1, i} = P_{g, 0}$
senão
se ($q_{i-1} < R_i \leq q_i$) **então**
$P_{g-1, i} = P_{g, i}$
fim-se
i = i + 1
até ($i \geq$ npop)
fim do método reprodução

inicio do método cruzamento ($P_{g, i-1}$, $P_{g, i}$) {operador fusion + 2 pontos de corte}
se (random() < 0.5) **então**
operador *fusion* da seção 3.6.4
senão
pcross1 = rnd() * n, pcross2 = rnd() * n
i = pcross1 + 1
repita
K = $P_{g, i-1}$
$P_{g, i-1} = P_{g, i}$
$P_{g, i} = K$
i = i + 1
até ($i \geq$ pcross2)
fim-se
fim do método cruzamento

inicio do método mutação (K, $P_{g, L}$) {alteração nos últimos 10% da população}
se (((g/maxgen)*100) < 90) **então**
se ($P_{g, L, K} = 1$) **então**
$P_{g, L, K} = 0$
senão
se ($P_{g, L, K} = 1$) **então**
$P_{g, L, K} = 0$
senão
$P_{g, L, K} = 1$
fim-se
fim do método mutação

inicio do método seleção (P_g) {seleciona o indivíduo com a "melhor" aptidão}
min ($f(P_{g, i})$), $0 \leq i <$ npop
fim do método seleção

fim do algoritmo

Vale relembrar que o PCC e o PCV são problemas de minimização, portanto, o pseudocódigo acima, referente ao wAG, apresenta-se sob esta hipótese.

4.3 Complexidade do wAG

A função de complexidade do tempo computacional do wAG é dada por:

$$f(m, n, npop, maxgen, px) = g(m, n, npop) + maxgen[r(m, n) + c(px, npop, m, n) + s(m, n)]$$

Onde:

- g – geração da população inicial, viabilidade e avaliação;
- r – reprodução;
- c – cruzamento;
- s – seleção/substituição.

Seja K = ⟨m, n, npop, maxgen, px⟩, a concatenação dos parâmetros de *f*. Assim:

$$f(K) = g(m, n, npop) + maxgen[r(m, n) + c(px, npop, m, n) + s(m, n)]$$

$$f(K) = npop.g(m,n) + maxgen[r(m, n) + px.npop.c(m, n) + s(m, n)]$$

$$f(K) = npop.(mn) + maxgen[(mn) + px.npop.(mn) + (mn)]$$

Como $0 < px < 1$, sendo $maxgen = c_1 n$, $npop = c_0$ e $c_2 = 1 + px.c_0 + 1$, tem-se:

$$f(K) = c_0.mn + c_1 n.[1.(mn) + px.c_0.(mn) + 1.(mn)]$$

$$f(K) = c_0.mn + c_1 n.(c_2.mn)$$

$$f(K) = c_0.mn + c_1 c_2.mn^2$$

$$\boldsymbol{O[f] = O[mn^2]}$$

Vale ressaltar que a taxa de mutação, pm , não foi utilizada no cálculo da complexidade do wAG devido à rapidez com que o procedimento de Mutação é executado, sendo, portanto, seu tempo computacional desprezível em relação aos demais.

4.4 Passo-a-passo do wAG

Para fortalecer os conceitos de Algoritmo Genético, segue-se um passo-a-passo de um Problema de Transporte retirado de VIANA(1998, p. 82 – Exemplo 2.20 – com adaptações).

Uma companhia locadora de automóveis se defronta com um problema de alocação resultante dos contratos de locação, que permitem que os automóveis sejam devolvidos em localidades diferentes daquelas onde originalmente foram alugados. Num dado instante há m = 2 agências (ORIGEM) com respectivamente 12 e 19 carros excedentes e n = 4 outras agências (DESTINO) necessitando de 9, 6, 7 e 9 carros. Os custos unitários C_{ij} , em

reais, de transporte entre as locadoras "i" e "j" são os seguintes: $C_{11} = 45$, $C_{12} = 17$, $C_{13} = 21$, $C_{14} = 30$, $C_{21} = 14$, $C_{22} = 18$, $C_{23} = 19$ e $C_{24} = 31$. Determine a forma de alocação dos carros entre as agências de forma a minimizar o custo de transporte.

O modelo original de VIANA (1998) não trás a variável X_{99} nem as desigualdades $\geq$ nas restrições. Na verdade, todas as restrições são de igualdade e a variável X_{99} não existe. Tal modelo foi relaxado, ou seja, adaptado para facilitar a execução do wAG. X_{ij} corresponde a quantidade de carros a serem transportados da agência "i" de origem (i = 1 ou 2) para a agência "j" de destino (j = 1, 2, 3 ou 4).

Modelo original:

Minimizar

$$\mathbf{45X_{11} + 17X_{12} + 21X_{13} + 30X_{14} + 14X_{21} + 18X_{22} + 19X_{23} + 31X_{24}}$$

Sujeito a

$$\boldsymbol{X_{11} + X_{12} + X_{13} + X_{14} = 12}$$
$$\boldsymbol{X_{21} + X_{22} + X_{23} + X_{24} = 19}$$
$$\boldsymbol{X_{11} + X_{21} = 9}$$
$$\boldsymbol{X_{12} + X_{22} = 6}$$
$$\boldsymbol{X_{13} + X_{23} = 7}$$
$$\boldsymbol{X_{14} + X_{24} = 9}$$

Modelo adaptado:

Minimizar

$$\mathbf{45X_{11} + 17X_{12} + 21X_{13} + 30X_{14} + 14X_{21} + 18X_{22} + 19X_{23} + 31X_{24} + 1000X_{99}}$$

Sujeito a

$$\boldsymbol{X_{11} + X_{12} + X_{13} + X_{14} + X_{99} \geqslant 12}$$
$$\boldsymbol{X_{21} + X_{22} + X_{23} + X_{24} + X_{99} \geqslant 19}$$
$$\boldsymbol{X_{11} + X_{21} + X_{99} \geqslant 9}$$
$$\boldsymbol{X_{12} + X_{22} + X_{99} \geqslant 6}$$
$$\boldsymbol{X_{13} + X_{23} + X_{99} \geqslant 7}$$
$$\boldsymbol{X_{14} + X_{24} + X_{99} \geqslant 9}$$

A solução ótima para o Problema de Transporte modelado acima é $X_{11} = 0$; $X_{12} = 6$; $X_{13} = 0$; $X_{14} = 6$; $X_{21} = 9$; $X_{22} = 0$; $X_{23} = 7$; $X_{24} = 3$, com um custo mínimo de R$ 634,00. Tanto o LINDO (1994) quanto o wAG encontram a mesma solução ótima, mesmo com as devidas adaptações.

Utilizou-se um vetor de matrizes de tamanho variável, para armazenar as populações que são geradas de acordo com o parâmetro repassado. É a partir da população inicial que o algoritmo é capaz de gerar as gerações seguintes. Os cromossomos são representados também por vetores de tamanho variável conforme Figura 6. As instâncias de testes advém de arquivos externos que são lidos pelo wAG pelo método leitura.

Conforme introduzido pela seção 3.5.2 e descrito pela 4.2, a geração da população inicial ocorre pela geração de cada cromossomo. A matriz P_0 é usada para armazenar a população inicial, onde cada linha representa um cromossomo e cada coluna um *gene* inteiro. O método geração, detalhado abaixo, é o responsável por gerar cada cromossomo e, consequentemente, a população inicial.

inicio do método geração (P_0) {gera a população inicial}
inicio da Heurística Gulosa com peso 1
para cada coluna j ∈ J **faça**
$P_{0,1,j} = 0$
enquanto I ≠ ∅ **faça**
$j \leftarrow argmin\left\{ k, \frac{1}{|A_k|}, |A_k| \neq 0 \right\}$
$P_{0,1,j} = 1$
Para cada linha i ∈ I **faça**
Se $a_{ij} = 1$, **então**
I = I – {i}
fim-enquanto
fim da Heurística Gulosa com peso 1

inicio da Heurística Gulosa com peso real
para cada coluna j ∈ J **faça**
$P_{0,2,j} = 0$
enquanto I ≠ ∅ **faça**
$j \leftarrow argmin\left\{ k, \frac{w_k}{|A_k|}, |A_k| \neq 0 \right\}$
$P_{0,2,j} = 1$
Para cada linha i ∈ I **faça**
Se $a_{ij} = 1$, **então**
I = I – {i},
fim-enquanto
fim da Heurística Gulosa com peso real

inicio da Heurística Aleatória
i = 3, S = ∅
repita
para cada coluna j ∈ J **faça**
$P_{0,i,j} = 0$

Selecione aleatoriamente uma coluna $j \in J$
Se $j \notin S$, **então**
$S \leftarrow S \cup \{j\}$
Para cada coluna $j \in J$ **faça**
$P_{0,i,j} = 1$
fim-se
$i = i + 1$
até $(i \geq npop)$
fim da Heurística Aleatória
fim do método geração

Os resultados obtidos estão representados na Tabela 2, que corresponde à população inicial , utilizando os seguintes parâmetros: npop = 20 (tamanho da população), maxgen = 100 (número máximo de gerações), px = 0,50 (percentual de cruzamento igual a 50%) e pm = 0,10 (percentual de mutação igual a 10%). Para este exemplo, não foi utilizada a mutação adaptativa devido ao pequeno número de cromossomos na população inicial.

TABELA 2: Configurações da população inicial obtida na execução do wAG.

i	Configuração C_i	$1 / f_i$	p_i (%)	q_i
1	(15,5,8,3,6,16,0,6,3)	0,0002185	9,02	0,09
2	(6,15,16,5,17,13,18,18,3)	0,0001858	7,67	0,17
3	(18,6,5,3,3,16,18,16,4)	0,0001594	6,58	0,23
4	(2,7,5,4,12,6,18,11,12)	0,0000747	3,09	0,26
5	(13,6,14,13,4,0,0,6,17)	0,0000537	2,22	0,29
6	(11,17,14,4,1,16,12,3,13)	0,0000675	2,79	0,31
7	(6,4,2,15,18,0,14,0,12)	0,0000749	3,09	0,34
8	(10,8,6,11,7,12,16,17,5)	0,0001391	5,75	0,40
9	(2,17,11,0,0,9,1,13,17)	0,0000550	2,27	0,42
10	(8,4,11,10,9,18,12,15,0)	0,0004757	19,65	0,62
11	(16,14,2,9,11,1,14,7,15)	0,0000591	2,44	0,65
12	(4,3,1,10,18,17,18,5,17)	0,0000537	2,22	0,67
13	(10,17,18,4,18,13,12,11,8)	0,0000972	4,01	0,71
14	(18,10,9,7,11,13,5,4,4)	0,0001671	6,90	0,78
15	(2,16,3,12,13,5,9,0,9)	0,0000978	4,04	0,82
16	(10,7,18,17,10,3,4,17,12)	0,0000702	2,90	0,85
17	(9,3,8,12,14,18,1,4,10)	0,0000859	3,55	0,88
18	(14,0,11,7,9,14,15,18,4)	0,0001589	6,56	0,95
19	(17,15,12,18,1,0,9,18,17)	0,0000511	2,11	0,97
20	(1,5,17,4,8,13,4,5,12)	0,0000758	3,13	1,00

Tomando como exemplo a configuração $C_1\,(15,\,5,\,8,\,3,\,6,\,16,\,0,\,6,\,3)$, que passou pelo teste de viabilidade, ou seja, satisfez todas as restrições, quando seus respectivos valores foram substituídos no função objetivo, o resultado obtido foi 0,0002185. Há de se ressaltar que, como o problema é de minimização, a coluna da função objetivo é $\frac{1}{f(C_i)}$, se fosse de maximização seria $f(C_i)$. Sabe-se que este resultado não é o melhor, por isso a necessidade de várias gerações e algumas perturbações nos cromossomos.

Para o cálculo das probabilidades de seleção p_i de cada elemento, bem como a probabilidade acumulada q_j , com $q_0=0$, mencionadas em seções anteriores, são representadas e implementadas pelo wAG através do pseudocódigo abaixo.

início
soma = 0
para i=1 **até** npop **faça**
    soma = soma + [1 / f($P_{g,i}$)]
para i=1 **até** npop **faça**
    p_i = [1 / f($P_{g,i}$)] / soma
para j=1 **até** n **faça**
    para i=1 **até** npop **faça**
        $q_j = q_j + p_i$
fim

Logo, para $i = 1$, temos, de forma abreviada:

$$p_1 = \frac{f_1}{f_1 + \ldots + f_{20}} = \frac{0{,}0002185}{0{,}0002185 + \ldots + 0{,}0000758} = 0{,}0902$$

Na fase de Reprodução, a "roleta" viciada, responsável por selecionar os indivíduos mais aptos da população, copia cada cromossomo proporcionalmente ao valor da função objetivo e o associa a uma fatia igual a sua aptidão, o método foi descrito na seção 4.2 da seguinte forma:

inicio do método reprodução (P_{g-1})
    i = 0
    repita
        R_i = rnd()
        se ($0 < R_i \leq q_0$) **então**
            $P_{g-1,\,i} = P_{g,\,0}$

```
            senão
                se (q_{i-1} < R_i ≤ q_i) então
                    P_{g-1, i} = P_{g, i}
                fim-se
            fim-se
            i = i + 1
        até (i ≥ npop)
fim do método reprodução
```

Com base na Tabela 2, foram gerados os seguintes valores para R_i, i = 1, 2, …, 20: 0,1831; 0,1058; 0,7847; 0,9302; 0,8384; 0,2048; 0,9689; 0,6572; 0,5318; 0,4760; 0,5649; 0,6127; 0,2631; 0,3646; 0,8484; 0,0489; 0,6534; 0,0457; 0,8097; 0,1279. Assim, a geração seguinte, C_i', i = 1, 2, …, 20, reproduzida pelo wAG, foram os cromossomos C_3, C_2, C_{15}, C_{18}, C_{16}, C_3, C_{20}, C_{12}, C_{10}, C_{10}, C_{10}, C_{10}, C_4, C_8, C_{17}, C_1, C_{12}, C_1, C_{15}, C_2. Observa-se que a tendência é a multiplicação dos indivíduos mais fortes: C_{10} que é o melhor foi quadruplicado; C_1, C_2 e C_3 que são bons, foram duplicados; e a eliminação dos mais fracos: C_{19}, que é o pior, foi eliminado; C_9, C_5 e C_{11}, que são ruins, também foram eliminados. O conceito de "melhor", aqui, refere-se ao menor valor da função objetivo, uma vez que o problema é de minimização. Os cromossomos "bons", referem-se àqueles cuja aptidão está mais próxima do melhor e os "ruins", caso contrário. Entretanto, percebe-se que o indivíduo C_{14}, que é bom, foi eliminado, enquanto que C_{12}, que é ruim, foi duplicado. As operações seguintes são exatamente para corrigir este imprevisto.

Outro fator importante a ser considerado é o Cruzamento. Uma vez que foi utilizado para a taxa de Cruzamento *px* o valor de 0,50, estima-se que 50% da população, em torno de 10 indivíduos, ou 5 cruzamentos, serão atingidos. O método cruzamento apresentado na seção 4.2 é detalhado abaixo:

```
inicio do método cruzamento (P_{g, i-1}, P_{g, i})        {operador fusion + 2 pontos de corte}
    se (rnd() < 0.5) então
        para cada coluna j ∈ J faça
            se (P_{g, i-1, j} = P_{g, i, j}) então
                C_j = P_{g, i, j}
            fim-se
            se (P_{g,i-1,j} ≠ P_{g,i,j}) então
                C_j = P_{g, i-1, j} com probabilidade p = f(P_{g, i}) / [f(P_{g, i-1})+ f(P_{g, i})]
                C_j = P_{g, i, j} com probabilidade 1-p
            fim-se
        fim-para
    senão
        pcross1 = rnd() * n
```

```
            pcross2 = rnd() * n
            i = pcross1 + 1
            repita
                  K =  P_{g, i-1}
                  P_{g, i-1} = P_{g, i}
                  P_{g, i} = K
                   i = i + 1
            até (i ≥ pcross2)
      fim-se
fim do método cruzamento
```

Conforme a Tabela 3, foram recombinados os cromossomos: C_1 com C_2, C_5 com C_6, C_9 com C_{10}, C_{17} com C_{18} e C_{19} com C_{20}. Os demais permaneceram inalterados. Com exceção do primeiro cruzamento, S_1 com S_2, que não sofreu alterações, em todos os outros houveram melhorias, representada na Tabela 3 pelas setas voltadas para baixo (↓), e para cima (↑), caso contrário. Inicialmente, a melhor aptidão era do cromossomo S_{10}, mas, após o cruzamento, houve uma redução significativa de 1893 para 1465, representado pelo cromossomo C_9.

TABELA 3: Configurações da população inicial pós-Cruzamento e ponto de corte.

i	Configuração C_i	f_i	pcross	Configuração C_i'	f_i'	Situação
1	(15,5,8,3,6,16,0,6,3)	4576	8	(15,5,8,3,6,16,0,6,**3)**	4576	inalterado
2	**(6,15,16,5,17,13,18,18,3)**	5383		**(6,15,16,5,17,13,18,18,**3)	5383	inalterado
5	(13,6,14,13,4,0,0,6,17)	18613	5	(13,6,14,13,4**,16,12,3,13)**	15036	↓
6	**(11,17,14,4,1,16,12,3,13)**	14821		**(11,17,14,4,1,**0,0,6,17)	18398	↑
9	(2,17,11,0,0,9,1,13,17)	18403	6	(2,17,11,0,0,9,**12,15,0)**	1465	↓
10	**(8,4,11,10,9,18,12,15,0)**	1893		**(8,4,11,10,9,18,**1,13,17)	18831	↑
17	(9,3,8,12,14,18,1,4,10)	11647	6	(9,3,8,12,14,18,**15,18,4)**	6347	↓
18	**(14,0,11,7,9,14,15,18,4)**	6292		**(14,0,11,7,9,14,**1,4,10)	11592	↑
19	(17,15,12,18,1,0,9,18,17)	19660	1	(17,**5,17,4,8,13,4,5,12)**	13904	↓
20	**(1,5,17,4,8,13,4,5,12)**	13079		**(1,**15,12,18,1,0,9,18,17)	18835	↑

Sendo a taxa de Mutação igual a 0,10, estima-se que 10% da população, ou seja, apenas 2 indivíduos serão atingidos. O pseudocódigo simplificado abaixo ilustra a maneira como o algoritmo se comporta durante a mutação de um gene.

inicio do método mutação (K, $P_{g, L}$)
$P_{g, L, K} = P_{g, L, K} - 1$

fim do método mutação

Observa-se que o algoritmo decrementa uma unidade no valor do gene K escolhido aleatoriamente, se houverem melhorias e o cromossomo resultante for viável, o mesmo é armazenado, caso contrário, conserva-se o gene original.

TABELA 4: Configurações da população inicial pós-Mutação.

i	Configuração C_i	f_i	gen	Configuração C_i'	f_i'	Situação
2	(6,15,16,5,17,13,18,**18**,3)	5383	8	(6,15,16,5,17,13,18,**17**,3)	5352	↓
20	(**1**,15,12,18,1,0,9,18,17)	18835	1	(**0**,15,12,18,1,0,9,18,17)	18790	↓

A Tabela 4 mostra os dois cromossomos escolhidos para a fase de Mutação, os genes que foram alterados e o resultado deste processo. Ela mostra claramente uma melhora na aptidão dos cromossomos que sofreram mutação: o 2º, na posição 8 e o 20º na 8ª posição.

Por fim, o wAG seleciona o melhor elemento, ou seja, aquele com a melhor aptidão, conforme indicado abaixo.

inicio do método seleção (P_g) {seleciona o indivíduo com a "melhor" aptidão}
L = 99999, i=0
repita
se ($f(P_{g,i}) < L$) **então**
$L = f(P_{g,i})$
$K = i$
fim-se
até ($i \geq npop$)
$C = P_{g,k}$
fim do método seleção

5 RESULTADOS COMPUTACIONAIS

Esta seção trata de apresentar os resultados computacionais obtidos durante a realização dos testes no wAG. Nas Tabelas 2 e 3, referentes ao Problema da Cobertura de Vértices (PCV) , as colunas n, m, Vo, To, Vg, Tg, Min, Med, Max, DvP, Rz e TwAG representam, respectivamente, o número de vértices/variáveis, o número de ligações/ restrições; o valor ótimo e o tempo ótimo – obtidos no LINDO (1994) –; o valor guloso e o tempo guloso – obtidos no algoritmo guloso em Viana (2006) –; o valor mínimo, a média de dez execuções do algoritmo, o valor máximo, o desvio padrão e o tempo médio de execução obtidos no AG. As instâncias testadas foram as seguintes: wvc01, wvc02, wvc03, wvc04, wvc05, wvc06, wvc07, wvc08, wvc09, wvc10, wvc11, wvc12, nesta ordem. Todas extraídas da Biblioteca de Testes em Pesquisa Operacional OR Library, (BEASLEY, 1990).

TABELA 5: Resultados do wAG para instâncias do PCV, sem peso.

Instância		**Ótimo**		**Guloso**		**wAG**					
n	**m**	**Vo**	**To**	**Vg**	**Tg**	**Min**	**Med**	**Max**	**DvP**	**Rz**	**TwAG**
800	1500	454	9m11s	674	1s	476	479,75	481	1,82	1,05	9s
800	2000	482	9m57s	686	1s	498	500,67	501	0,89	1,03	11s
1000	2000	588	10m13s	842	1s	611	612,58	613	0,79	1,04	14s
1000	2500	601	10m49s	846	1s	630	632,42	633	1,00	1,05	17s
1250	2500	716	11m07s	1054	1s	751	751,83	752	0,39	1,05	22s
1500	3000	864	13m14s	1277	1s	921	923,33	924	0,98	1,07	31s
1750	3000	972	14m17s	1470	1s	1033	1034,80	1035	0,62	1,06	38s
2000	3500	1140	17m28s	1678	1s	1188	1189,60	1190	0,79	1,04	52s
2250	3500	1225	18m35s	1908	1s	1302	1306,00	1307	1,71	1,06	1m2s
2500	3500	1250	19m55s	2112	1s	1427	1431,30	1432	1,48	1,14	1m8s
2750	3500	1375	21m32s	2334	1s	1549	1569,80	1572	6,59	1,14	1m20s
3000	4000	1595	23m27s	2546	1s	1717	1721,20	1722	1,75	1,08	1m40s
			14m59s	**Média total das instâncias**						1,07	59s

O número de iterações do AG foi reduzido, visando a diminuição do tempo de execução do algoritmo. Os parâmetros utilizados para as instâncias do PCV, sem peso, foram: 100 cromossomos na população inicial, 15 gerações/iterações, 50% para a taxa de Cruzamento e 10% para a taxa de Mutação. Observa-se que as colunas Min e Tag possuem valores menores que os das colunas Vg e Tg de Viana (2006). A Figura 10 representa o gráfico do mesmo conjunto de dados da Tabela 5, fazendo uma comparação entre os tempos de cada instância testada do PCV.

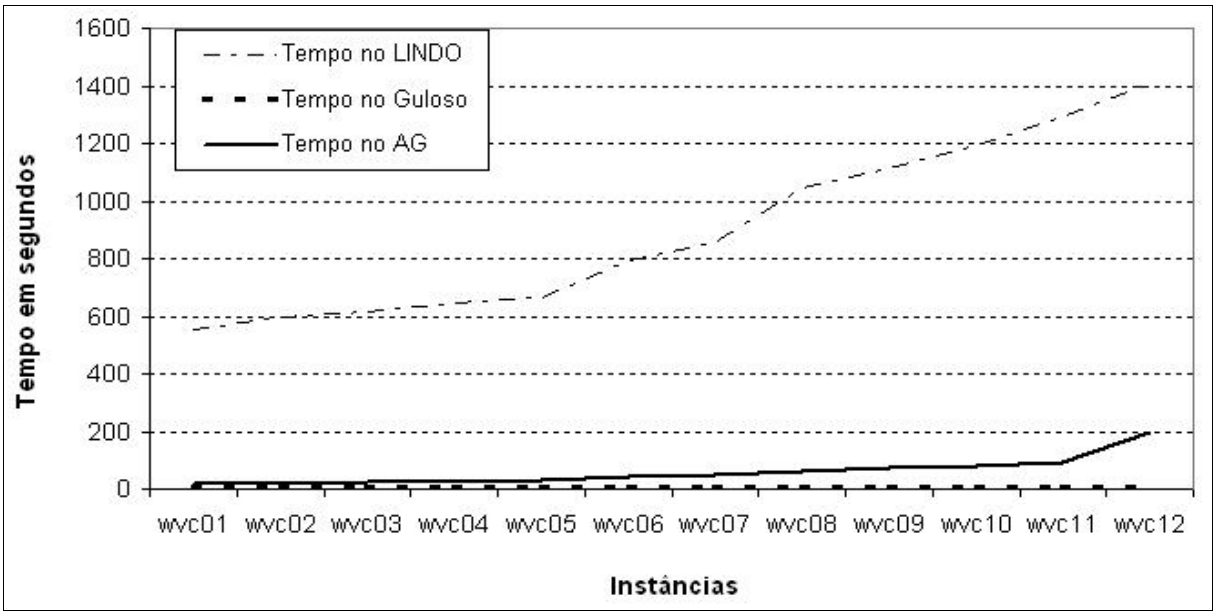

FIGURA 10: Gráfico comparativo dos tempos das instâncias do PCV, sem peso.

Pela Figura 10, observa-se que o tempo no AG foi melhor que o do LINDO (1994), para instâncias sem peso. A Figura 11 abaixo, representa o gráfico do mesmo conjunto de dados da Tabela 5, fazendo uma comparação entre os melhores resultados obtidos de cada instância testada do PCV.

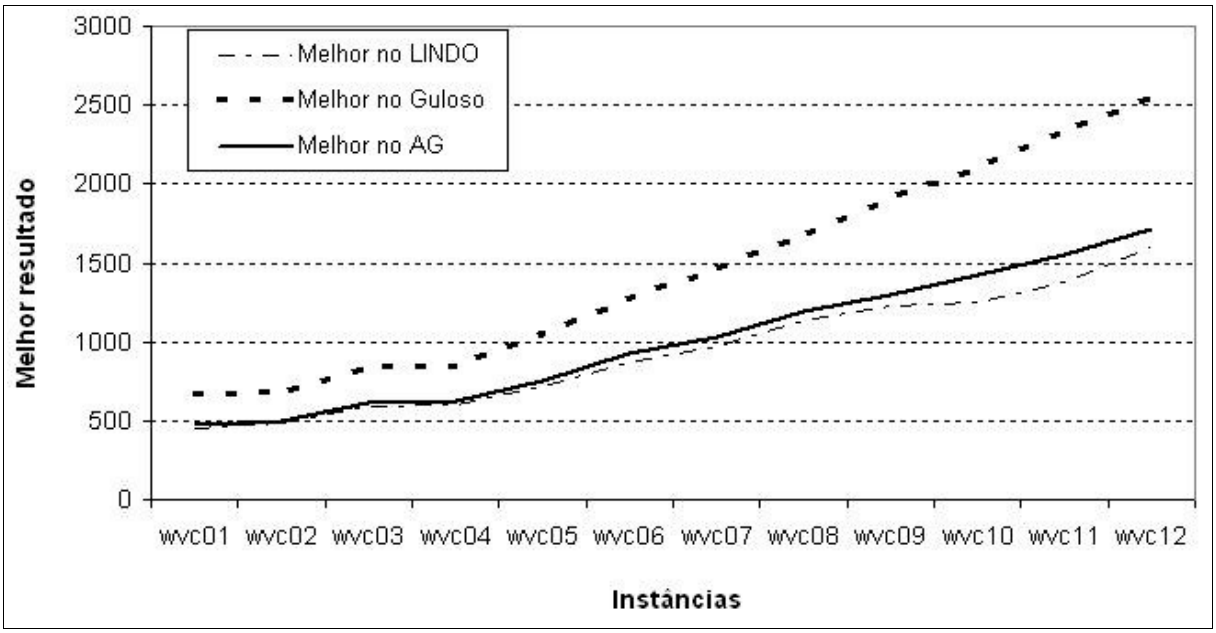

FIGURA 11: Gráfico comparativo dos melhores resultados das instâncias do PCV, sem peso.

Pela Figura 11, observa-se que o resultado no AG foi melhor que o do algoritmo guloso de Viana (2006) e chegou bem próximo do resultado do LINDO (1994), e isso em um tempo bem menor representado pela Figura 10.

TABELA 6: Resultados do wAG para instâncias do PCV, com peso.

Instância		Ótimo		Guloso		wAG					
n	m	Vo	To	Vg	Tg	Min	Med	Max	DvP	Rz	TwAG
800	1500	19627	38m	33398	1s	20120	20576	20618	143,73	1,03	28s
800	2000	20751	40m	33238	1s	21658	21811	21825	48,21	1,04	33s
1000	2000	25073	42m	42431	1s	26514	26642	26627	58,36	1,06	45s
1000	2500	27259	43m	43731	1s	28061	28229	28263	79,21	1,03	53s
1250	2500	31705	44m	52741	1s	33431	33598	33615	52,80	1,05	1m11s
1500	3000	38317	46m	64953	1s	40617	40850	40871	73,32	1,06	1m42s
1750	3000	43194	49m	76115	1s	45606	45783	45811	64,03	1,06	2m8s
2000	3500	48265	50m	84481	1s	50828	51047	51119	131,16	1,05	2m46s
2250	3500	52775	51m	95832	1s	55981	56169	56186	59,18	1,06	3m17s
2500	3500	55904	52m	105013	1s	59340	59451	59461	34,90	1,06	3m48s
2750	3500	60635	53m	115804	1s	63838	63939	63982	65,50	1,05	4m20s
3000	4000	67673	56m	128756	1s	71562	71799	71821	74,77	1,06	5m25s
			47m	**Média total das instâncias**						1,05	2m16s

Os parâmetros utilizados para as instâncias do PCV, com peso, foram: 100 cromossomos na população inicial, 50 gerações/iterações, 50% para a taxa de Cruzamento e 10% para a taxa de Mutação. A Figura 12 representa o gráfico do mesmo conjunto de dados da Tabela 6, fazendo uma comparação entre os tempos de cada instância testada do PCV.

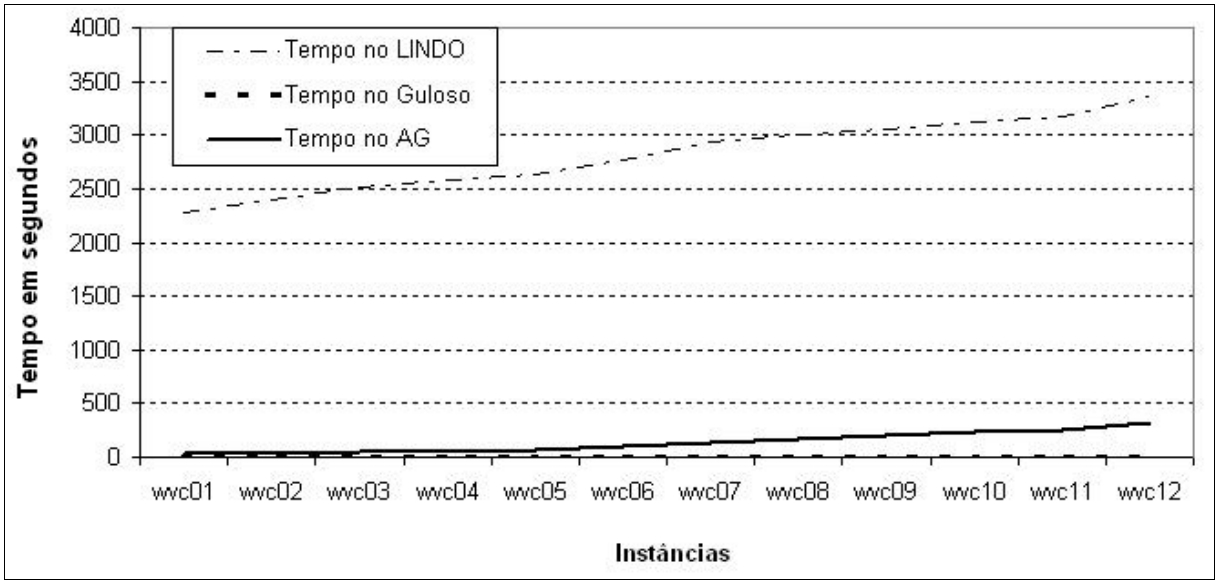

FIGURA 12: Gráfico comparativo dos tempos das instâncias do PCV, com peso.

Pela Figura 12, observa-se que o tempo no AG também foi melhor que o resultado do LINDO (1994), para instâncias com peso. A Figura 13 representa o gráfico do mesmo conjunto de dados da Tabela 6, fazendo uma comparação entre os melhores resultados obtidos de cada instância testada do PCV.

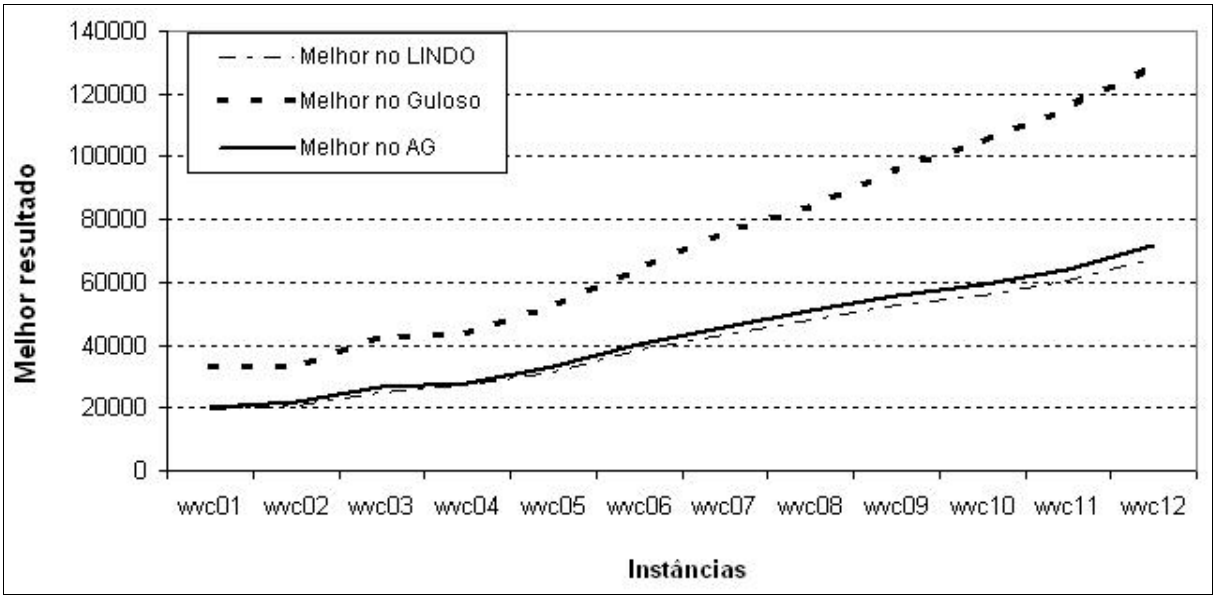

FIGURA 13: Gráfico comparativo dos melhores resultados das instâncias do PCV, com peso.

De acordo com a Figura 13, observa-se que o resultado no AG também foi melhor que o do algoritmo guloso de Viana (2006) e chegou bem próximo do resultado ótimo do LINDO (1994), e isso em um tempo bem menor representado na Figura 12.

TABELA 7: Resultados do wAG para instâncias do PCC, sem peso.

Instância		Ótimo		Alg. Guloso		wAG					
m	**n**	**Vo**	**To**	**Vg**	**Tg**	**Min**	**Med**	**Max**	**DvP**	**Rz**	**TwAG**
200	1000	39	3m39s	41	1s	41	42,3	45	2,08	1,05	2s
200	2000	35	4m18s	37	1s	37	36,5	39	1,52	1,05	9s
300	3000	40	4m38s	42	1s	42	44,1	44	1,15	1,05	38s
400	4000	40	7m01s	47	1s	47	48,5	50	1,52	1,17	57s
240	192	60	*	**	**	60	60	60	0	1,00	1s
672	448	144	*	148	1s	147	149,8	151	1,52	1,02	1s
1792	1024	344	*	364	1s	361	365,1	368	2,08	1,04	6s
4608	2304	780	*	816	1s	816	817,4	821	2,64	1,04	36s
11520	5120	1792	*	1969	1s	1925	1927,7	1928	1,52	1,07	3m18s
28160	11264	4103	*	**	**	4304	4306,9	4309	2,51	1,05	17m18s
			*	**Média total das instâncias**						1,05	2m18s

As instâncias testadas foram as seguintes: scp41, scp51, scpa1, scpc1, scpcyc06, scpcyc07, scpcyc08, scpcyc09, scpcyc10, scpcyc11, nesta ordem. Todas extraídas da mesma Biblioteca de Testes em Pesquisa Operacional OR Library, (BEASLEY, 1990) anteriores.

Na Tabela 7, os asteriscos simples (*) simbolizam que o LINDO (1994) demorou mais de 1 hora – tempo limite – para encontrar o resultado ótimo e, por isso, não foram computados aqui, razão pela qual a Figura 14 possui uma reta subindo indefinidamente; os

asteriscos duplos (**) significam que os resultados de tais instâncias não foram apresentados pelo artigo de Viana et al (2006).

O número de iterações do AG também foi aqui reduzido, visando a diminuição do tempo de execução do algoritmo. Os parâmetros utilizados para as instâncias do PCC, sem peso, foram: 10 cromossomos na população inicial, 15 gerações/iterações, 50% para a taxa de Cruzamento e 10% para a taxa de Mutação. Entretanto, se o tamanho da população e o número de gerações aumentarem, os resultados serão melhores, como é o caso da instância *scpcyc07*, cujo m=672, n=448, teve como melhor resultado 145 após 21 minutos e 38 segundos de execução; e da *scpcyc08*, cujo m=1792, n=1024, teve como melhor resultado 348 após 7 horas e 15 minutos. A Figura 14, abaixo, representa o gráfico do mesmo conjunto de dados da Tabela 7, fazendo uma comparação entre os tempos de cada instância testada do PCC.

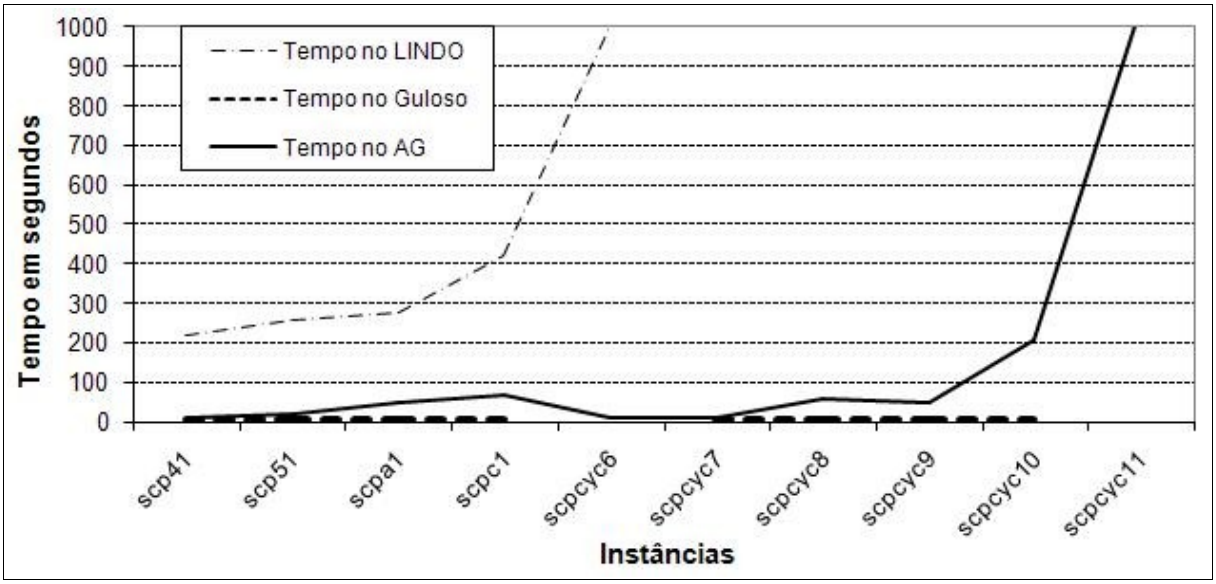

FIGURA 14: Gráfico comparativo dos tempos das instâncias do PCC, sem peso.

Pela Figura 14, observa-se que o tempo no AG foi melhor que o do LINDO (1994) – que ultrapassou o tempo limite –, para instâncias sem peso. A Figura 15 representa o gráfico do mesmo conjunto de dados da Tabela 7, fazendo uma comparação entre os melhores resultados obtidos de cada instância testada do PCC.

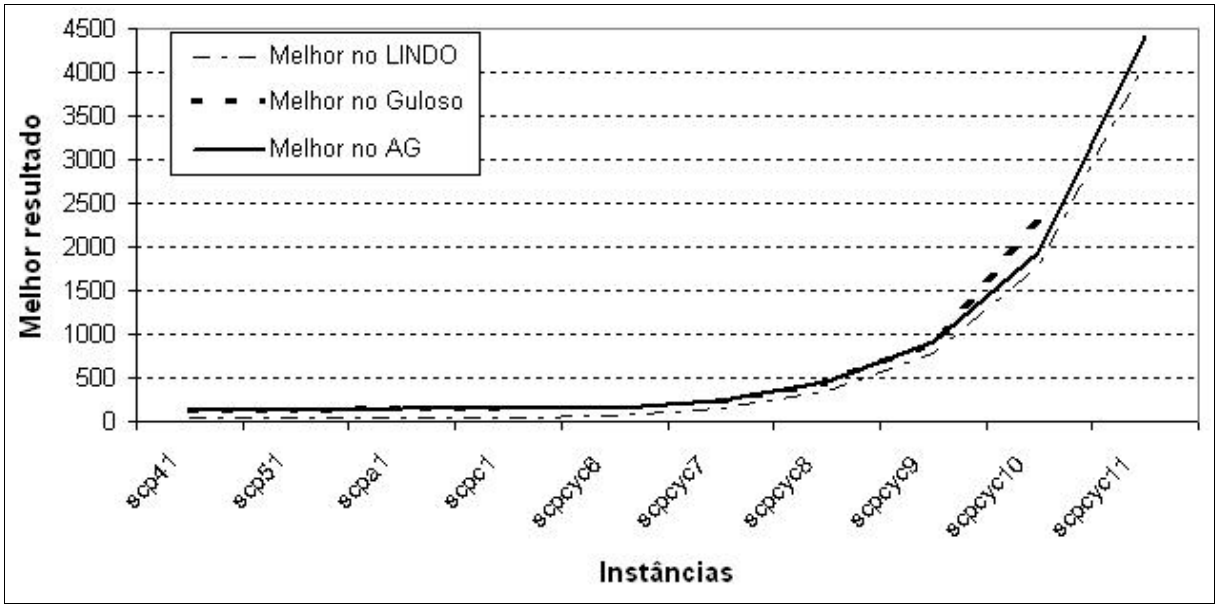

FIGURA 15: Gráfico comparativo dos melhores resultados das instâncias do PCC, sem peso.

Pela Figura 15, observa-se que o resultado no AG foi um pouco melhor que o do algoritmo guloso de Viana et al (2006) e chegou bem próximo do resultado do LINDO (1994), e isso em um tempo bem menor representado pela Figura 14.

TABELA 8: Resultados do wAG para instâncias do PCC, com peso.

Instância		Ótimo		Alg. Guloso		wAG					
m	**n**	**Vo**	**To**	**Vg**	**Tg**	**Min**	**Med**	**Max**	**DvP**	**Rz**	**TwAG**
200	1000	429	**	463	1s	437	1678,2	3416	1533,97	1,02	3s
200	2000	253	**	293	1s	286	2398,0	3454	1559,82	1,13	11s
300	3000	253	**	288	1s	285	2479,5	4047	1937,16	1,12	33s
400	4000	227	**	261	1s	250	1619,6	4359	2023,13	1,10	1m1s
240	192	60	**	**	**	60	60,0	60	0,00	1,00	1s
672	448	144	**	148	1s	147	149,8	151	1,52	1,02	1s
1792	1024	344	**	364	1s	361	365,1	368	2,08	1,04	6s
4608	2304	780	**	816	1s	816	817,4	821	2,64	1,04	36s
11520	5120	1792	**	1969	1s	1925	1927,7	1928	1,52	1,07	3m18s
28160	11264	4103	**	**	**	4304	4306,9	4309	2,51	1,05	17m18s
			**				**Média total das instâncias**			1,06	2m18s

Os parâmetros utilizados para as instâncias do PCC, com peso, foram: 10 cromossomos na população inicial, 15 gerações/iterações, 50% para a taxa de Cruzamento e 10% para a taxa de Mutação. A Figura 16 representa o gráfico do mesmo conjunto de dados da Tabela 8, fazendo uma comparação entre os tempos de cada instância testada do PCC.

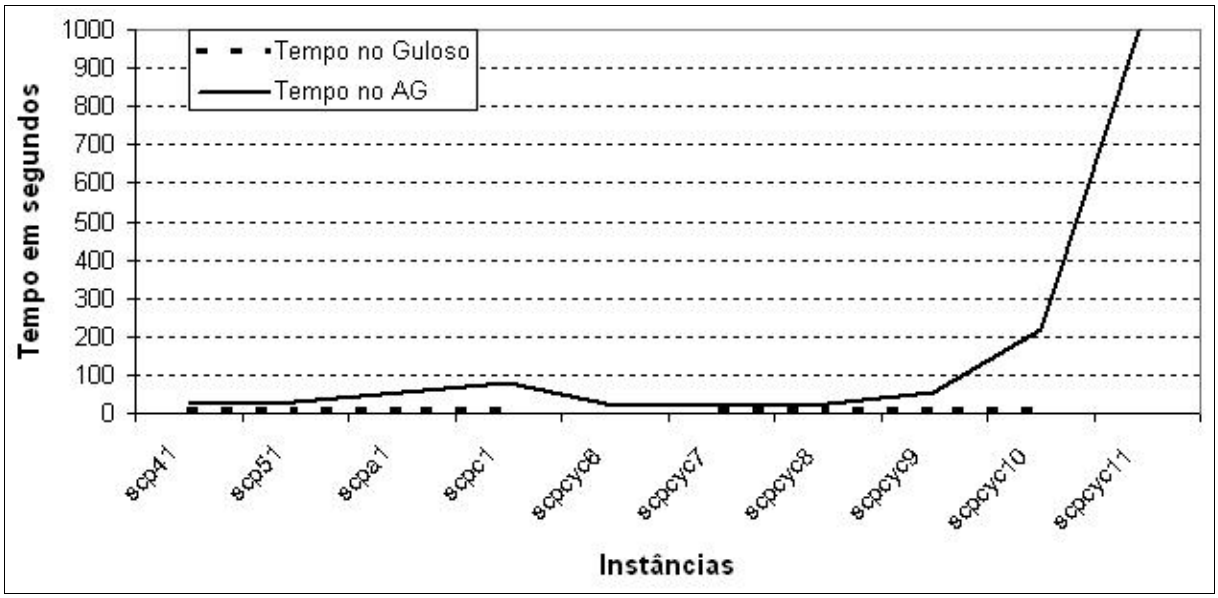

FIGURA 16: Gráfico comparativo dos tempos das instâncias do PCC, com peso.

A diferença entre o tempo de execução para instâncias com e sem peso é quase nula. A Figura 17 representa o gráfico do mesmo conjunto de dados da Tabela 8, fazendo uma comparação entre os melhores resultados obtidos de cada instância testada do PCC.

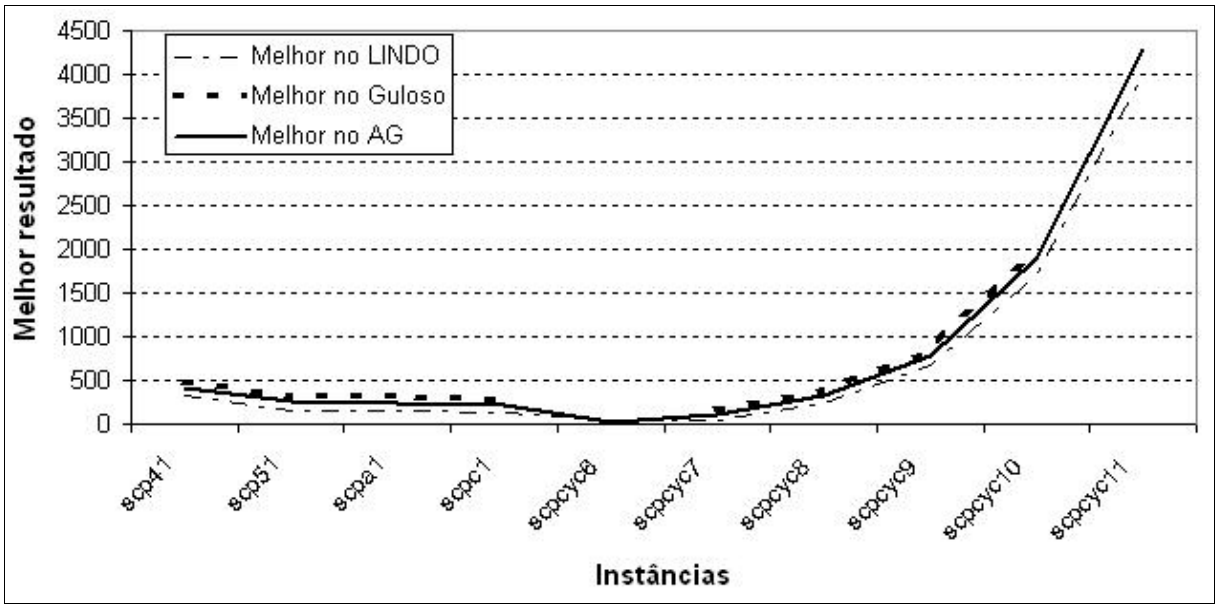

FIGURA 17: Gráfico comparativo dos melhores resultados das instâncias do PCC, com peso.

Pela Figura 17, observa-se que o resultado no AG também foi melhor que o do algoritmo guloso de Viana et al (2006) e chegou bem próximo do resultado ótimo do LINDO (1994).

6 CONCLUSÃO

Pelos testes realizados, o ótimo de instâncias com um pequeno número de variáveis e restrições é rapidamente alcançado pelo wAG. Para instâncias com um alto número de variáveis e restrições, o wAG encontrou um resultado bem próximo do ótimo, com um tempo computacional maior, porém bem menor que os *solvers* tradicionais. Bons resultados são também obtidos quando se aumenta o número de iterações, mas isto também prejudica grandes instâncias. O tempo comprometido e este prejuízo nem se comparam ao resultado encontrado por ambientes já consagrados, pois, como exemplo, para a instância scpcyc06 - o ótimo, cujo valor é igual a 60, foi encontrado no LINGO (2008) em 20 horas. O wAG encontrou o mesmo valor em 1 segundo. Verificou-se, ainda, que o agrupamento de técnicas de Cruzamento, como o operador *fusion* e o uso de dois pontos de corte, 50% para cada técnica, bem como o uso de 10% para a taxa de mutação final na Mutação Adaptativa, ao invés dos 1% proposto pela literatura, favoreceram a melhoria do tempo de convergência do wAG. Outro fator que contribuiu significativamente e está no princípio do algoritmo é a qualidade da população inicial, que, além de ser gerada aleatoriamente, foi gerada também por meio de uma Heurística Gulosa com base nas restrições.

Acrescenta-se, ainda, a melhoria dos resultados após a substituição de apenas um dos bits no operador Mutação na maioria das iterações do algoritmo. Os testes computacionais comprovaram que tal procedimento produz uma busca profunda na vizinhança do espaço de soluções depois de o wAG já ter diversificado bem através do operador Cruzamento. Por tudo isso, constata-se a contribuição deste trabalho para a comunidade científica, em ter melhorado os resultados de Viana et al(2006), sendo o wAG um algoritmo mais eficiente para encontrar boas soluções para problemas de Cobertura de Conjuntos e de Vértices. Em muitas ocasiões, o wAG encontrou resultados melhores que o algoritmo guloso proposto por Viana et al (2006).

Trabalhos subsequentes poderão desenvolver: o estudo de um ponto de percentagem ótimo para a substituição de apenas um dos bits no operador Mutação; uma aplicação mais amigável através de ambientes computacionais que permitam a interação de estudantes da área de otimização, objetivando a aprendizagem destes.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVIN, A. C. de F. **Uma heurística híbrida de melhoria para o problema de bin-packing e sua aplicação ao problema de escalonamento de tarefas**. Tese (Doutorado em Informática) – Pontifícia Universidade Católica do Rio de Janeiro. 2003.

BAKER, E. K. **Heuristic Algorithms for the Weighted Set Covering Problem**. *Computers and Operations Research*, 8(4), 303-310. 1981.

BALAS, E.; HO, A. **Set covering algorithms using cutting planes, heuristics and subgradient optimization: a computational study**. Mathematical Programming Study 12: 37-60, 1980.

BEASLEY, J. E. **OR-Library**. Journal of the Operational Research Society 41, 1990.

BRANCO, I. M.; COELHO, J. D. **P-mediana hamiltoniana: resolução heurística**. Lisboa: [s.n]. 1984.

CASTRO, E. C. **Pré-processamento do problema de cobertura de conjuntos aplicado ao escalonamento de condutores**. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Maringá, 2006.

CHONG, E. K. P. e ZAK, S. H. **An introdution to optimization.** 2. ed. New York: Wiley. 2001.

CHU, P. **A genetic algorithm approach for combinatorial optimization problems**. PhD thesis, The Management School, Imperial College of Science, Technology and Medicine, London, 1997.

CHU, P. e Beasley, J. E. **A genetic algorithm for the set covering problem**. European Journal of Operational Research, 94: 392-404, 1996.

CHVÁTAL, V. **A greedy heuristic for the set covering problem**. Mathematics of Operations Research, 4: 233-235, 1979.

CORMEN, T. H. et al. **Algoritmos: teoria e prática**. Tradução. 2. ed. Rio de Janeiro: Campus, 2002.

FISHER, M.L.; KEDIA, P. **Optimal solution of set covering/partitioning problems using dual heuristics**. Management Science 36 (1990) 674-688, 1990.

HARCHE, F.; THOMPSON, G.L. The column subtraction algorithm: An exact method for solving weighted set covering, packing and partitioning problems. Computers & Operations Research 21 (1994) 689-705, 1994.

HONG, T. P., WANG, H. S., CHEN, W. C. **Simultaneously Applying Multiple Mutation Operators in Genetic Algorithms**. Journal of Heuristics, 6: 439-455, 2000.

JENSEN, P. A. e BARD, J. F. **Operations research models and methods**. New York: Wiley. 2003.

LEVINE, D. **A parallel genetic algorithm for the set partitioning problem**. PhD thesis, Department of Computer Science, Illinois Institute of Technology, 1994.

LINDO. versão 5.3 para PC. Chigaco: Lindo Systems Inc., 1994.

LINGO – Optimization Software. Versão 11. Chigaco: Lindo Systems Inc, 2008.

LOPES, L. de S. **Uma heurística baseada em algoritmos genéticos aplicada ao problema da cobertura de conjuntos**. Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, 1995.

LUDESCHER, L. G.; PEREZ, R. A. A. P.; MATOS, T. **Algoritmo genético e programação genética**. São Paulo: [s.n], 2008.

MAHEY, P. **Programação não-linear: uma introdução à teoria e aos principais métodos**. Rio de Janeiro: Campus, 1987.

MARTÍNEZ, J. M. e SANTOS, S. A. **Métodos computacionais de otimização**. Campinas: [s.n.] 1995.

NEPOMUCENO, N. V. **Combinação de Meta-heurísticas e Programação Linear Inteira: uma Metodologia Híbrida Aplicada ao Problema de Carregamento de Contêiner**. Dissertação (Mestrado em Informática) - Universidade de Fortaleza, 2006.

NOVO Michaelis: **dicionário ilustrado**. 28. ed. São Paulo: Melhoramentos, 1998. Português.

OLIVEIRA, A. C. M. de. **Introdução à pesquisa operacional**. Maranhão: [s.n.]. [200-]. Apostila.

OLIVEIRA, D. G. **Estudo comparativo entre metaheurísticas populacionais com tamanho da população variável**. Dissertação (Mestrado em Informática) - Universidade de Fortaleza, 2008.

OLIVEIRA. N. V. **Problema de Cobertura de Conjuntos – uma comparação numérica de algoritmos heurísticos**. Dissertação (Mestrado em Engenharia de Produção) – Universidade Federal de Santa Catarina, 1999.

RODRIGUES, F. L. et al. **Metaheurística algoritmo genético para solução de problemas de planejamento florestal com restrições de integridade**. Revista Árvore, Viçosa, v. 28, n. 2, 2004.

SANTOS, F. J.; VIANA, G. V. R.; THOMAZ, A. C. F. **Algoritmo para seleção de carteiras de investimentos com risco mínimo.** Revista Científica da Faculdade Lourenço Filho, Fortaleza, CE, v.1, p.4, n.2. 2002.

SILVA, E. L.; MENEZES, E. M. **Metodologia da pesquisa e elaboração de dissertação**. 3. ed. rev. atual. Florianópolis: Laboratório de Ensino a Distância da UFSC, 2001.

SOARES, G. L. **Algoritmo Genético: Estudo, Novas Técnicas e Aplicações**. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Minas Gerais, 1997.

TAHA, H. **Pesquisa Operacional**. 8. ed. São Paulo: Pearson Prentice Hall, 2008.

VASKO, F.J.; WILSON, G.R. **An efficient heuristic for large set covering problems**. Naval Research Logistics Quarterly, 31: 163-171, 1984.

VIANA, G. V. R. **Meta-heurísticas e programação paralela em otimização combinatória**. Fortaleza: UFC, 1998. 250p.

VIANA, G. V. R et al. **Experimental Analysis of Approximation Algorithms for the Vertex Cover and Set Covering Problems**. Computers & Operations Research 33 3520–3534, 2006.

WILLIAMS, H. P. **Model building in mathematical programming**. 4. ed. New York: Wiley, 1999.

ANEXO: Código java do wAG

```
import java.io.*;
import java.text.DecimalFormat;
import java.util.Vector;

public class AG_PLI {

	static AG_PLI ag_pli = new AG_PLI();
	static Vector<String> linha_do_arquivo,restricoes;
	static Vector<Integer> aux = new Vector<Integer>();
	static String fo,desigualdade;
	static int numRes,numVar,ngens,melhor_f,npop,ti,maxgen,k,l,pcross1,pcross2;
	static int []cromo,f,melhor_cromo,d,e,w;
	static int [][] pop0,matriz_ct;
	static double px,pm,pf,n;
	static double []p,q,r;
	static DecimalFormat ap= new DecimalFormat("0");
	static Vector<int[]> matriz_ct = new Vector<int[]>();
	static Vector<Integer> kk = new Vector<Integer>();
	static Vector<Integer> peso = new Vector<Integer>();

	public static void main(String[] args) {
		leitura("scp41-200x1000.txt");
		parametros(100,50,0.5,0.1)
		gerar_pop_inicial();
		selecao(pop0);
		System.out.println("melhor_f da pop inicial: "+melhor_f);
		n = 2.3025;
		pf = 0.1;

		for(int g=0; g < maxgen; g++){
			reproducao(pop0);
			if(usar_crossover= =true){
			for (int i=0;i<ag_pli.getNpop();i+=2){
				if(Math.random()<=ag_pli.getPx()){//if1
					for(int j = 0 ; j<d.length; j++){
						d[j] = pop0[i][j];
						e[j] = pop0[i+1][j];
					}
					cruzamento(d,e);
					if (viavel(d))
						for(int j=0;j<d.length;j++)
							pop0[i][j]= d[j];
					if (viavel(e))
						for(int j=0;j<e.length;j++)
							pop0[i+1][j]= e[j];
				}
				selecao(pop0);
```

```
                }
                }
                if(usar_mutacao= =true){
                if (usar_mutacao_adaptativa= =true){
                        ag_pli.setPm(pf*Math.exp(n*(g-maxgen)/maxgen));
                        if(g= =maxgen-1) ag_pli.setPm(pf);
                }
                for (int i=0;i<ngens;i++){
                        if(Math.random()<=ag_pli.getPm()){
                                l = (int) Math.floor(i/numVar);
                                k = i - l*numVar;
                                for(int j=0; j<d.length; j++)
                                        d[j] = pop0[l][j];
                                e = mutacao(k,d,g);
                                if (viavel(e))
                                        for(int j=0;j<e.length;j++)
                                                pop0[l][j]= e[j];
                        }
                }
                }
        }
        imprimir();
}
public static void leitura(String arquivo){
        linha_do_arquivo = new Vector<String>();
        restricoes = new Vector<String>();
        ag_pli.setMelhor_f(99999);
        acionar_peso = true;
        try {
                AG_PLI.carregar(arquivo);
                } catch (Exception e) {
                        e.printStackTrace();
                }
}
public static void carregar(String arquivo)
throws FileNotFoundException, IOException {
File file = new File(arquivo);
        if (! file.exists())
                System.out.println("O arquivo não existe!");
        BufferedReader br = new BufferedReader(new FileReader(arquivo));
        String linha;
        while( (linha = br.readLine()) != null ){
                linha_do_arquivo.add(linha);
        }
        br.close();
}
public static void parametros(int npop, int maxgen, double px, double pm){
        ag_pli.setNpop(npop);
        ag_pli.setMaxgen(maxgen);
        ag_pli.setPx(px);
```

```
        ag_pli.setPm(pm);
}
public static void gerar_pop_inicial(){
        ngens = npop*numVar;
        cromo = new int[numVar];
        melhor_cromo = new int[numVar];
        pop0 = new int[npop][numVar];
        f = new int[npop];
        p = new double[npop];
        q = new double[npop];
        r = new double[npop];
        d = new int[numVar];
        e = new int[numVar];
        for(int i=0; i<npop; i++){
                construir3();
                for(int j=0; j<numVar;j++)
                        pop0[i][j]=cromo[j];
        }
        if(usar_guloso= =true){
                guloso();
                System.out.print("cromo guloso gerado em ");
                for(int j=0; j<numVar;j++)
                        pop0[0][j]=cromo[j];
        }
        calc(pop0);
}
public static void guloso(){
        Vector<Integer> cardinalidade = new Vector<Integer>();
        for (int i=0; i<numVar;i++){
                cardinalidade.add(0);
                cromo[i]=0;
        }
        int w;
        for (int i=0; i<matriz_ct.size();i++){
                for(int j=0; j<matriz_ct.get(i).length; j++){
                        w = cardinalidade.get(matriz_ct.get(i)[j]-1);
                        cardinalidade.removeElementAt(matriz_ct.get(i)[j]-1);
                        cardinalidade.add(matriz_ct.get(i)[j]-1,w+1);
                }
        }
        while (matriz_ct.size()!=0){
                int n=0;
                if(acionar_peso==true){
                        double m=(double)peso.get(10)/cardinalidade.get(0);
                        for (int i=0; i<cardinalidade.size(); i++){
                                if ((double)peso.get(i)/cardinalidade.get(i)<m){
                                        m=(double)peso.get(i)/cardinalidade.get(i);
                                        n=i;
                                }
                }
```

```
        }else{
                double m=(double)1/cardinalidade.get(0);

                for (int i=0; i<cardinalidade.size(); i++){
                        if ((double)1/cardinalidade.get(i)<m){
                                m=(double)1/cardinalidade.get(i);
                                n=i;
                        }
                }
        }
        int i=0;
        while(i<matriz_ct.size()){
                for(int j=0; j<matriz_ct.get(i).length; j++){
                        if(matriz_ct.get(i)[j]==n+1){
                                for(int jj=0; jj<matriz_ct.get(i).length; jj++){
                                    w = cardinalidade.get(matriz_ct.get(i)[jj]-1);
                        cardinalidade.removeElementAt(matriz_ct.get(i)[jj]-1);
                        cardinalidade.add(matriz_ct.get(i)[jj]-1,w-1);
                }
                                matriz_ct.remove(i);
                                i=-1;
                                break;
                        }
                }
                i++;
        }
        cromo[n]=1;
        cardinalidade.removeElementAt(n);
        cardinalidade.add(n,0);
}
String b="";
int []t;
for(int x2=0; x2<restricoes.size(); x2++){
        for(int x3=0; x3<restricoes.get(x2).length(); x3++){
                if(restricoes.get(x2).charAt(x3)=='X'){
                        while(restricoes.get(x2).charAt(x3+1)!=' '){
                                b+=restricoes.get(x2).charAt(x3+1);
                                x3++;
                        }
                        kk.add(Integer.parseInt(b));
                        b="";
                }
        }
        b="";
        t = new int[kk.size()];
        for(int k1=0; k1<t.length; k1++)
                t[k1]=kk.get(k1);
        matriz_ct.add(t);
        kk.removeAllElements();
}
```

```
}
public static void construir(){
	int r;
	Vector<Integer> aux = new Vector<Integer>();
	Vector<Integer> aux2 = new Vector<Integer>();
	for(int i=0; i<cromo.length; i++)
		cromo[i]=0;
pai:for(int i=0; i<matriz_ct.size(); i++){
		for(int j=0; j<matriz_ct.get(i).length; j++){
			aux.add(cromo[matriz_ct.get(i)[j]-1]);
			aux2.add(matriz_ct.get(i)[j]);
		}
		for(int k=0; k<aux.size(); k++)
			if(aux.get(k)= =1)
				continue pai;
		r = (int) (Math.random()*matriz_ct.get(i).length);
		cromo[matriz_ct.get(i)[r]-1]=1;
		aux2.remove(r);
		for(int l=0; l<aux2.size(); l++)
			if(cromo[aux2.get(l)-1]==0)
				cromo[aux2.get(l)-1]=-1;
		aux.removeAllElements();
		aux2.removeAllElements();
	}
	for(int m=0; m<cromo.length; m++)
		if(cromo[m]= =-1)
			cromo[m]=0;
	if(!viavel(cromo)){
	  pai:for(int i=0; i<matriz_ct.size(); i++){
			for(int j=0; j<matriz_ct.get(i).length; j++)
				if(cromo[matriz_ct.get(i)[j]-1]==1)
					continue pai;
			r = (int) (Math.random()*matriz_ct.get(i).length);
			cromo[matriz_ct.get(i)[r]-1]=1;
	  }
	}
}
public static boolean viavel(int...c){
	pai:for(int i=0; i<matriz_ct.size(); i++){
		for(int j=0; j<matriz_ct.get(i).length; j++)
			if(c[matriz_ct.get(i)[j]-1]==1)
				continue pai;
		return false;
	}
return true;
}
public static int aptidao(int...c){
	int soma=0;
	for(int i=0; i<c.length; i++)
		soma+=c[i]*peso.get(i);
```

```
        return soma;
}
public static void selecao(int [][] g){
        for(int i=0;i<g.length;i++)
                if(f[i] < ag_pli.getMelhor_f()){
                        ag_pli.setMelhor_f(f[i]);
                        for(int j=0;j<g[i].length;j++)
                                melhor_cromo[j]=g[i][j];
                }
}
public static void reproducao(int[][] m){
        for(int i=0;i<m.length;i++){
                r[i]=Math.random();
                if ((0<r[i])&(r[i]<=q[0])){
                        for(int j=0;j<m[i].length;j++)
                                pop0[i][j]= m[0][j];
                }else{
                        for(int j=1;j<m.length;j++)
                                if ((q[j-1]<r[i])&(r[i]<=q[j]))
                                        for(int w=0;w<m[i].length;w++)
                                                pop0[i][w]=m[j][w];
                }
        }
        calc(pop0);
}
public static int[] mutacao(int gen, int[] vetor,int g){
        int h = (g/ag_pli.getMaxgen())*100;
        if(h<90){
                if (vetor[gen]= =1)
                        vetor[gen]=0;
        }else{
                if (vetor[gen]= =1){
                        vetor[gen]=0;
                }else{
                        vetor[gen]=1;
                }
        }
        return vetor;
}
public static void cruzamento(int[] a, int[] b){
        double r = Math.random();
        if(r<0.5){
                int []filho1,filho2;
                filho1 = new int[numVar];
                filho2 = new int[numVar];
                double []pdf = new double[2];
                pdf[0] = (double) aptidao(b)/(aptidao(a)+aptidao(b));
                pdf[1]= 1.0 - pdf[0];
                for (int i = 0; i < numVar; i++){
                        if (a[i] == b[i]){
```

```
                        filho1[i] = a[i];
                        filho2[i] = a[i];
                    }else{
                        int s = roleta(pdf);
                        if (s == 1){
                            filho1[i] = a[i];
                            filho2[i] = b[i];
                        }else{
                            filho1[i] = b[i];
                            filho2[i] = a[i];
                        }
                    }
                }
                ag_pli.setD(filho1);
                ag_pli.setE(filho2);

            }else{
                int xxx, yyy;
                xxx = (int) (Math.random()*numVar-1);
                yyy = (int) (Math.random()*numVar-1);
                pcross1 = Math.min(xxx,yyy);
                pcross2 = Math.max(xxx,yyy);
                int aux;
                for(int i=pcross1+1; i<pcross2; i++){
                    aux = a[i];
                    a[i]= b[i];
                    b[i]=aux;
                }
                ag_pli.setD(a);
                ag_pli.setE(b);
            }
        }
        public static int roleta (double []pdf){
            double t = Math.random();
            if(t<=pdf[0])
                return 1;
            return 0;
        }
        public static void calc(int[][] m){
            double soma=0;
            for(int j=0; j<m.length; j++)
                f[j]=0;
            for(int i=0; i<m.length; i++)
                for(int j=0; j<m[i].length; j++)
                    f[i] += m[i][j];
            for(int i=0;i<f.length;i++)
                soma += (double) 1/f[i];
            for(int j=0; j<m.length; j++)
                p[j]=0;
            for(int i=0;i<p.length;i++)
```

```
                p[i]=((double)1/f[i])/soma;
            for(int j=0; j<m.length; j++)
                q[j]=0;
            for(int j=0;j<q.length;j++)
                for(int i=0;i<=j;i++)
                    q[j]+=p[i];
        }
        public static void imprimir(){
            System.out.print("\nmelhor_f: "+melhor_f);
            if(viavel(melhor_cromo)){
                System.out.print(", viável, em ");
            }else{
                System.out.print(", inviável, em ");
            }
            stime();
            for(int j=0; j<melhor_cromo.length; j++)
                System.out.print(melhor_cromo[j]+" ");
        }
}//FIM DO ALGORITMO
```