UNIVERSIDADE FEDERAL DE MINAS GERAIS

Curso de Pós Graduação em Engenharia Metalúrgica e de Minas

Dissertação de Mestrado

"Aplicação de algoritmos genéticos na determinação de cava final e sequenciamento de lavra em minas a céu aberto"

Autor: Octávio Rosa de Almeida Guimarães

Orientadora: Professora Maria de Fátima Gripp

Fevereiro/2007

UNIVERSIDADE FEDERAL DE MINAS GERAIS
Curso de Pós Graduação em Engenharia Metalúrgica e de Minas

Octávio Rosa de Almeida Guimarães

APLICAÇÃO DE ALGORITMOS GENÉTICOS NA DETERMINAÇÃO DE CAVA FINAL E SEQÜENCIAMENTO DE LAVRA EM MINAS A CÉU ABERTO

Dissertação de Mestrado apresentada ao Curso de Pós-Graduação em Engenharia Metalúrgica e de Minas da Universidade Federal de Minas Gerais

Área de concentração: Tecnologia Mineral
Orientadora: Professora Maria de Fátima Gripp

Belo Horizonte
Escola de Engenharia da UFMG
Fevereiro/2007

Folha de aprovação

A meus pais,
pela educação e oportunidade.
A minha esposa Lu,
pelo apoio e paciência.
A meus filhos Ana Cristina e Vinicius,
pelas alegrias e inspirações.

AGRADECIMENTOS:

O autor agradece a todos aqueles que, direta ou indiretamente, colaboraram na preparação deste trabalho e, em particular:

À Professora Maria de Fátima Gripp pela competência, clareza e objetividade na orientação deste estudo.

Aos professores da UFMG Ricardo Takahashi e João Vasconcelos pelos esclarecimentos durante as aulas de otimização multiobjetivo e pelas opiniões na elaboração dos algoritmos implementados.

Aos professores do Departamento de Engenharia de Minas da UFMG Antonio Carlos Girodo, George Eduardo Sales Valadão e ao amigo Beck Nader pela indicação junto ao Centro de Pós Graduação em Engenharia Metalúrgica e de Minas - CPGEM.

Aos demais professores e colaboradores do Departamento de Engenharia de Minas e Metalúrgica da UFMG, bem como aos colaboradores do CPGEM, pela atenção, apoio e contribuição a minha formação profissional.

À Companhia Vale do Rio Doce e à Gemcom do Brasil, nas pessoas dos Geólogos Marcelo Rossi e Hildegundes Silva, e dos Engenheiros Fabiano Araújo, Maurílio Botelho, Paulo Sachs e Marcos Rittner, pela oportunidade e apoio.

Aos colegas de profissão Charles Henrique Sales Valadão, Josimar Souza Pires, Joaquim Pedro de Toledo e Antonio Carlos Girodo pela motivação e inspiração.

Ao Matemático Computacional Alexandre Marinho pelas sugestões de programação e auxílio na codificação dos programas.

À amiga Solange Nobre pelo interesse, disponibilidade, atenção e ajuda na revisão final.

SUMÁRIO

Lista de Figuras

Lista de tabelas

RESUMO

Os tópicos descritos baseiam-se em desenvolvimentos e técnicas divulgadas e têm por objetivo avaliar a aplicabilidade dos algoritmos genéticos no problema de definição de cava final e seqüenciamento de lavra para minas a céu aberto.

O objetivo é desenvolver um conjunto de rotinas computacionais baseado nos algoritmos genéticos que, a partir de um modelo de blocos tecnológicos conhecido representando uma jazida, seja capaz de gerar um conjunto de soluções ótimas, tendo em vista os objetivos de maximização de valor, respeitando as condicionantes de qualidade (teores) e distribuição das massas no tempo (seqüenciamento de lavra).

Foram desenvolvidos três programas que abordam a teoria dos algoritmos genéticos. O primeiro calcula os limites de cava final, a partir de um modelo de blocos visando a otimização do lucro não descontado, por meio de programa de computador baseado no algoritmo genético simples com elitismo. O segundo é um programa baseado no algoritmo genético multiobjetivo NSGA (*Non-dominated Sorting Genetic Algorithm*), que foi inicialmente aplicado a dois problemas multiobjetivos disponíveis na literatura e, em seguida, na otimização do lucro não descontado e para o problema multiobjetivo de maximização de reservas e do lucro. O terceiro, além de conter os algoritmos implementados anteriormente, comporta outros métodos de seleção, mutação, heurísticas de agrupamento e geração da população inicial.

Todos os programas implementados resolveram os problemas até então simulados de maneira eficaz para um número reduzido de blocos.

ABSTRACT

Genetic Algorithms are stochastic search techniques based on natural selection and genetic principles.

This work illustrates how genetic algorithms can be applied to optimize the final pit and scheduling of open pit mines.

Three computer programs were developed. The first one calculates the final pit based on a simple genetic algorithm with elitism maximizing the undiscounted pit value. The second was based on the multi-objective method NSGA (*Non-dominated Sorting Genetic Algorithm*), which was applied to two multi-objective classical problems, the final pit problem maximizing the pit value and the multi-objective problem of maximizing the pit value and the resources. The third one have more selection and mutation methods and was added some cluster and initial population heuristics

All computer programs solved the formulated problems for small block models.

1 Introdução e Objetivos

Os limites finais da cava definem o tamanho e a forma de uma mina a céu aberto no final de sua vida útil e, geralmente, buscam a maximização do lucro. Eles definem a extensão das reservas lavráveis e a quantidade de material estéril a ser retirado e depositado. Normalmente marcam a fronteira-limite além da qual a explotação de um dado depósito não será mais econômica. Os limites da cava na superfície delimitam uma fronteira dentro da qual as estruturas de superfície da mina, tais como plantas de beneficiamento e escritórios da mina, não devem ser locados.

Um dos problemas freqüentemente enfrentados por geólogos e engenheiros de minas é o da definição dos limites do corpo mineral assim como a avaliação da quantidade e da qualidade dos parâmetros de interesse. Existe uma série de métodos disponíveis para definir os limites de um dado corpo mineral. O método mais utilizado atualmente é a representação por um modelo de blocos (Kim, 1978), dividindo-se o corpo mineral em um conjunto de blocos. Saydam e Yalcin (2002) comentam que o planejamento de lavra baseado em um modelo de blocos envolve a decisão se um bloco do modelo deve ser lavrado ou não. Em caso afirmativo, quando o mesmo será lavrado e, uma vez extraído, quando deverá ser enviado ao processo. As respostas para cada um desses itens abordados, quando combinadas no contexto global do modelo de blocos, definem a progressão anual da cava e o fluxo de caixa advindo das operações mineiras (Dagdalen, 2001).

A figura a seguir apresenta um modelo conceitual de blocos. Segundo Halatchev (2002), a produção de uma seqüência otimizada de uma cava é um procedimento que reflete as condições de explotação ao longo da vida útil da mina. Essas condições são determinadas pelas características geológicas do depósito, condições de lavra e tecnologias de processamento mineral, além dos parâmetros econômicos. Do ponto de vista tecnológico, a seqüência ótima de lavra está diretamente relacionada com dois aspectos – espaço e tempo.

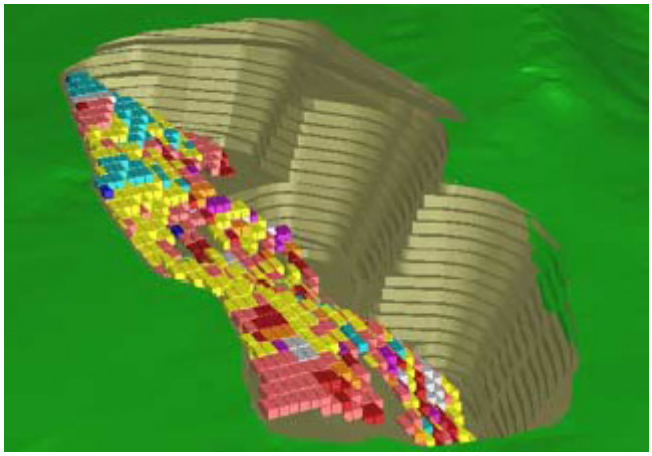


Figura 1.1 – Representação de um modelo de blocos.

Outro aspecto importante é que a seqüência ótima de produção da mina, em princípio, é usada como base para implementação de uma estratégia de teor de corte no contexto dos desenvolvimentos recentes. Dessa maneira, a otimalidade do teor de corte pode ser alcançada somente de maneira interativa considerando a otimalidade da seqüência de produção (Lane, 1988).

Dentro da concepção de otimização, que pode visar máxima lucratividade, maior valor presente líquido e aproveitamento dos recursos minerais, existe uma série de algoritmos desenvolvidos que se propõem a atingir tais objetivos porém os métodos que alcançaram a maior popularidade e conseqüente implementação computacional foram a técnica dos cones flutuantes (Pana & Carlson, 1966; David et al., 1974; Lemieux, 1979) e o algoritmo de Lerchs-Grossmann. (Lerchs e Grossmann, 1965).

Desde sua introdução, em 1965, o algoritmo de Lerchs-Grossmann (LG) é reconhecido por fornecer a solução ótima para o problema de projeto de cava a céu aberto. Recentemente, vários pesquisadores têm desenvolvido algoritmos e métodos alternativos para solucionar principalmente o problema de desempenho computacional do algoritmo original de LG, particularmente para problemas de grande número de blocos. Entre esses, autores como Zhao e Kim (1992), introduziram um novo algoritmo de projeto, baseado também na teoria dos grafos, reduzindo significativamente o número de arcos comparado com o algoritmo de LG. Huttagosol e Cameron (1992) apresentaram uma formulação para o problema de modelo de transporte, comparando os resultados obtidos com as abordagens de Ford e Fulkerson (1956) e os métodos de LG e cones flutuantes.

A geração de um projeto de cava final tem sido o primeiro passo de um planejamento de produção. Como resultado, muitos algoritmos de projeto de cava são desenvolvidos, tanto na categoria dos métodos heurísticos, método dos cones flutuantes (Lemieux, 1979), na categoria dos métodos matemáticos, tal como o método de Lerchs-Grossmann (Lerchs e Grossmann, 1965; Zhao e Kim, 1991). Invariavelmente esses algoritmos contam com modelos de blocos e, para cada bloco, é atribuído um valor monetário líquido (de ganho ou de perda). Desde que o valor do bloco seja uma função do preço do minério e dos custos de processo, a cava projetada com esse conjunto de valores fixos tornar-se-á obsoleta com o passar do tempo, quando os preços ou os custos sofrerem mudanças. Para acomodar as mudanças nos valores dos blocos, a "análise dos limites da cava" é praticada. Nessa análise, os parâmetros econômicos são sistematicamente modificados, um de cada vez, e uma cava é projetada após cada mudança.

A saída da análise de projeto é uma série de cavas onde cada uma possui seu próprio potencial de ser minerada, sob condições econômicas específicas. De fato, a análise de projetos de cava nada mais é do que uma parametrização respeitando os parâmetros econômicos (Wang & Sevim, 1995).

Diferentes métodos são usados para o projeto de limites finais de cava. A simulação e a programação dinâmica são as técnicas mais utilizadas. As técnicas de simulação incluem a técnica dos cones móveis. Os Métodos de programação dinâmica incluem algoritmos bidimensionais e tridimensionais (Carmo, 2001).

De acordo com Takahashi (2004), os algoritmos genéticos são caracterizados pela "evolução" de um conjunto de soluções-tentativas (população), segundo regras estocásticas de busca e combinação que leva de uma população à seguinte, numa seqüência de gerações. A existência de três regras básicas ou operadores genéticos básicos, de um Algoritmo Genético:

i. Um operador de cruzamento, que combina a informação contida em dois ou mais indivíduos (ou seja, duas ou mais soluções-tentativas), assim gerando outros indivíduos;

ii. Um operador de mutação que, utilizando a informação contida em um indivíduo, estocasticamente gera outro indivíduo; e

iii. Um operador de seleção que, utilizando a avaliação da função objetivo sobre todos os indivíduos da população, produz réplicas de alguns desses indivíduos e elimina outros indivíduos, assim gerando a população seguinte.

Um algoritmo genético pode ser construído a partir dessas três regras ou pode conter outros tipos de regras, tais como: nicho, busca local, etc.

Por meio dos algoritmos a serem desenvolvidos, pretende-se resolver os problemas propostos:

a) Determinação da cava final maximizando o lucro não descontado;
b) Determinação da cava final maximizando o lucro não descontado com restrições de qualidade (teores);
c) Determinação da cava final maximizando o lucro com restrições de qualidade e massas no tempo (seqüenciamento);
d) Geração das famílias de cavas ótimas visando maximizar recursos e lucro não descontado;
e) Geração das famílias de cavas ótimas visando maximizar recursos e teor médio;
f) Geração das famílias de cavas ótimas visando maximizar o lucro não descontado e teor médio.

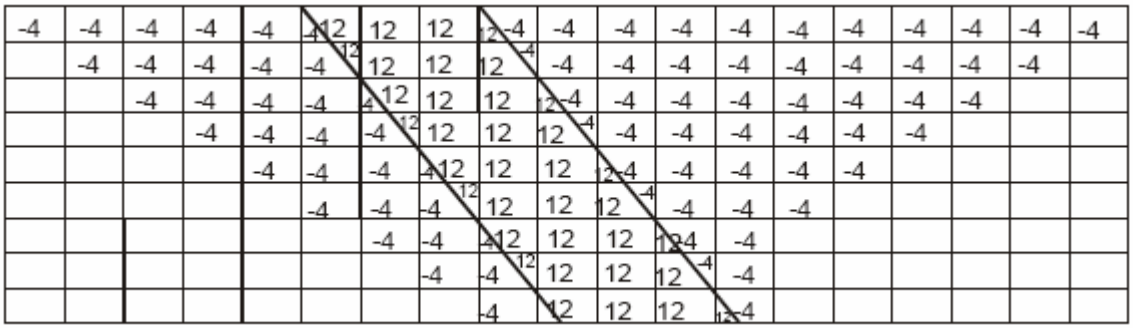
2 Revisão Bibliográfica

2.1 Descrição sumária de métodos de otimização de cava final

2.1.1 *O algoritmo de Lerchs-Grossmann*

Lerchs, H. e Grossmann, I. (1965), utilizando a técnica de programação dinâmica, desenvolveram, juntamente com um algoritmo de otimização bidimensional de cavas, um tratamento algébrico para a discretização da jazida em blocos tecnológicos. Um algoritmo derivado da teoria dos Grafos, trata o problema por meio da procura do fecho máximo em um grafo associado. A partir do benefício associado de lavra de um bloco *i*, representado por *B*i, o benefício global pode ser otimizado como a busca da combinatória de blocos que maximizem a somatória de *B*i respeitando as restrições pertinentes ao estudo.

A maneira mais simples de apresentar o método de Lerchs-Grossmann 2D é pelo uso de um exemplo (Peroni, 2002). Supondo um corpo mineral, onde os blocos estéreis tenham um valor (custo) de –4.000 unidades monetárias/bloco, e os blocos de minério apresentem um valor de 12.000 unidades/bloco, e ainda um ângulo de talude de 45 graus. Para cada bloco são atribuídos valores baseados em parâmetros econômicos conforme a figura abaixo:



adaptado de (Lerchs & Grossmann, 1965)

Figura 2.1.1.1 – Modelo de blocos inicial valorizado economicamente.

Como pode se observa, os blocos do limite do corpo mineral pertencem, tanto ao domínio do minério, quanto ao domínio do estéril. Um valor médio foi utilizado conforme a configuração apresentada a seguir:

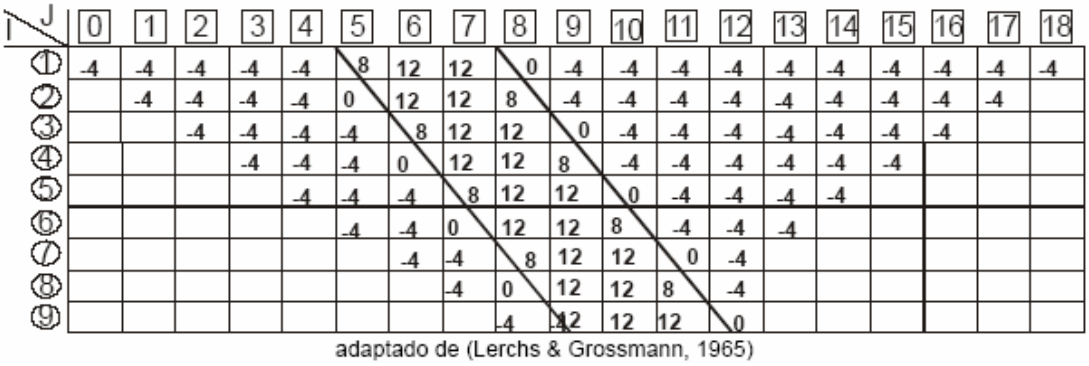


Figura 2.1.1.2 – Modelo de blocos econômico final.

A posição dos blocos será denotada utilizando-se a notação (*i,j*) para posicionar espacialmente os blocos, onde *i* representa a linha e j a coluna em que um determinado bloco está localizado.

Por meio de operações de soma nas colunas existentes no modelo de blocos e rotinas de procura de valores máximos, pode-se calcular o limite da cava final conforme figura a seguir:

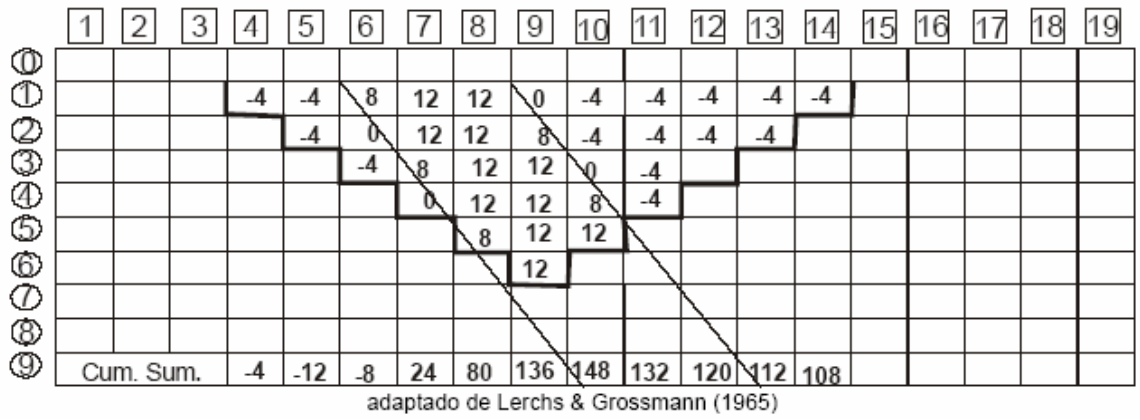


Figura 2.1.1.3 – Limite otimizado superposto ao modelo de blocos.

Uma evolução do algoritmo bidimensional de Lerchs-Grossmann foi proposta por Johnson & Sharp (1971). Essa modificação é referida em Barnes (1982) como o algoritmo 2 ½ D, pois considera um modelo tridimensional de blocos e analisa seção a seção em 2D até encontrar o contorno da cava final.

2.1.2 *Método dos cones flutuantes*

O método é baseado na pesquisa do contorno ótimo da cava final por tentativas. O princípio de funcionamento do algoritmo considera as restrições físicas e geomecânicas locais (ângulos de talude). O ápex do cone é movido de um bloco para outro e a avaliação do cone é feita em cada posição explorada. Quando a avaliação é positiva (cones fortes) o cone é selecionado e todos os blocos nele contido são extraídos. O processo repete-se até que não existam mais cones economicamente lavráveis. Esse método tem a vantagem de ser rápido e possui apelo bastante intuitivo (Underwood & Tolwinski, 1998). Yamatomi et al. (1995) apresentaram variantes dessa técnica.

Noronha (2001) afirma que esse método tem boa aplicação em corpos porfiríticos em que se tem um corpo central de minério e as encaixantes de estéril.

2.1.3 *Parametrização técnica de reservas*

Baseados em estudos de Matheron, G. (1975), Bongarçon, F. e Marechal, A. (1976) utilizaram a técnica de Análise Convexa de Vallet, R. (1976) para tratarem o problema de otimização de cava final por meio de uma aproximação funcional.

Conforme descrito por Noronha (2001), a idéia é baseada em voltar o problema para a determinação de uma função de parametrização técnica (função do teor médio dos blocos), a partir da qual torna-se possível a obtenção imediata de uma família de cavas ótimas, independente da conjuntura econômica subjacente ao problema. Em outras palavras, o conhecimento de todos os projetos potencialmente ótimos do ponto de vista de maximização da quantidade de metal com a minimização da remoção de material, ou seja, projetos ótimos do ponto de vista técnico, permitindo a comparação dos mesmos, com antecedência, às flutuações de parâmetros econômicos, por exemplo. A parametrização técnica de reservas, em resumo, é a procura dos projetos que pertençam à superfície convexa que se sobrepõe ao conjunto de todas as cavas possíveis.

2.2 Aplicação de algoritmos genéticos no planejamento de mina

A primeira utilização específica no planejamento de mina do Algoritmo Genético foi proposto por Tolwinski e Underwood (1992) e combina conceitos de otimização estocástica com conceitos de redes neurais para estabelecer um teor de corte ótimo no sentido de maximizar o valor econômico da jazida. Da mesma forma e adaptando a idéia do Algoritmo Genético, G. S. Thomas (1996) descreveu, em artigo específico, o conceito e a aplicação de um programa visando estabelecer uma seqüência ótima de lavra para uma mina a céu aberto (Toledo, 2003).

No Brasil, Toledo (2003) estudou duas aplicações de algoritmos genéticos. A primeira mostra a técnica da krigagem adaptativa cujo objetivo é selecionar um modelo variográfico (tipo, patamar, alcance e anisotropias) que minimize o erro da validação cruzada entre as amostras. Os resultados comprovaram a aderência do método em populações de comportamento conhecido. A segunda aplicação foi na seleção de projetos mineiros concorrentes, buscando, em sistemas de muitas minas e várias opções de investimentos, a combinação mais atrativa. A utilização do Algoritmo Genético solucionou o problema com precisão e agilidade e apresentou resultados melhores do que as abordagens tradicionais de otimização por programação linear. Portanto o Algoritmo Genético é uma ferramenta que poderá solucionar os problemas de combinações para seqüenciamento de lavra.

Durante o APCOM de 2005, Wageningen, Dunn & Muldowney analisaram a utilização de algoritmos genéticos combinados com Busca Tabu e simulação discreta de eventos no planejamento de mina de longo prazo. Segundo os autores a abordagem utilizada mostrou-se eficaz na resolução de problemas multiobjetivos e na geração de seqüenciamentos de lavra.

No mesmo evento, Samanta e Bhattacherjee (2005) avaliaram a aplicação de algoritmos genéticos no problema de otimização e controle de teores (Al_2O_3 e SiO_2), em um depósito de bauxita, gerando várias seqüências de extração de minério no tempo. Segundo os autores, a utilização de tal abordagem possibilitou a geração de vários cenários de lavra, permitindo, ao tomador de decisão, escolher, em função de suas necessidades, a operacionalização dos mesmos.

Outro exemplo de aplicação de algoritmos genéticos foi realizado por Thomas (1966) que ilustra sua aplicação e a utilização de técnicas de “recozimento” simulado

(*Simulated Annealing*) na determinação dos limites da cava final. O autor ressalta, como principal vantagem na utilização de algoritmos genéticos, a grande quantidade de soluções quase ótimas geradas, o que não acontece nas abordagens tradicionais. E, como principal desvantagem, o maior custo computacional exigido para atingir as soluções desejadas.

2.3 Otimização Multiobjetivo

Otimização Multiobjetivo, também chamada de otimização multicritério, multiperformance ou otimização vetorial pode ser definida como o problema de encontrar (Osyczka, 1985):

> "um vetor de variáveis de decisão que satisfaça um conjunto de restrições e otimize um vetor de funções cujos elementos representem a função objetivo. Estas funções formam a descrição matemática do chamado critério de desempenho, as quais usualmente são conflitantes. Assim, pode-se dizer que o termo otimizar significa descobrir uma solução para a qual os valores de todas as funções objetivo são considerados aceitáveis pelo projetista".

Formalmente, pode-se descrever o problema como se segue:

Encontrar um vetor $x^* = [x_1^*, x_2^*, ... x_n^*]$ que satisfaça as "*m*" restrições de desigualdade:

$$g_i(x) \leq 0 \quad \forall \quad i = 1, 2, ..., m \tag{2.3.1}$$

e as "*p*" restrições de igualdade

$$h_i(x) = 0 \quad \forall \quad i = 1, 2, ..., p \tag{2.3.2}$$

de modo a otimizar o seguinte vetor de funções:

$$f(x) = [f_1(x), f_2(x), ..., f_n(x)]^T \tag{2.3.3}$$

onde o vetor $x = [x_1, x_2, ... x_n]^T$ corresponde ao vetor de variáveis de decisão ou de otimização.

Ou seja, deseja-se determinar, sobre o conjunto *F* de todos os números que satisfaçam as restrições de igualdade e desigualdade, um conjunto particular de variáveis $x_1^*, x_2^*, ..., x_n^*$ que permitam atingir os valores eficientes de todas as funções objetivos baseados em algum fator de decisão.

O conjunto Pareto-Ótimo foi formulado por Pareto no século XIX e constitui a origem das pesquisas envolvendo otimização multiobjetivo (Pareto, 1896).

Um ponto x* pertencente ao conjunto viável *F* é Pareto-Ótimo se $\forall$ x* $\in$ *F* pode-se afirmar que:

$\exists$ i $\in$ [1, 2, ...,*n*] / $f_i(x^*) \leq f_i(x)$ e pelo menos uma dimensão j tal que $f_j(x^*) < f_j(x)$.

Em outras palavras, a definição diz que x* é Pareto ótimo se não existe nenhum vetor possível x que possa decrescer algum objetivo sem causar um acréscimo simultâneo em pelo menos outro objetivo.

O Pareto-Ótimo sempre gerará não apenas uma única solução, mas um conjunto de soluções chamadas não-inferiores, não-dominadas ou eficientes. O ponto $f^* = [f_1^*, f_2^*, ... f_n^*]$ é usualmente conhecido como utopia, ponto ideal ou ponto utópico. Esse ponto é impossível de ser alcançado, na maioria dos problemas multiobjetivos com objetivos conflitantes. Quando o ponto utópico pertence ao espaço viável de soluções, é obviamente a melhor solução e nenhuma pesquisa futura será necessária (Horn, 1997).

2.4 Elementos de um algoritmo genético

O Algoritmo Genético Simples - AGS, desenvolvido por Holland em seus trabalhos iniciais (Holland, 1975), foi o primeiro algoritmo genético inventado e é o protótipo no qual os outros se baseiam. Os algoritmos genéticos simples são importantes para o desenvolvimento teórico e têm grande valor didático, pois sua estrutura é a base sobre a qual as variações que definem os algoritmos mais avançados e recentes são desenvolvidas. Por isso, far-se-á uma descrição geral dos algoritmos genéticos simples e de seu funcionamento, baseada principalmente no livro de Goldberg (Goldberg, 1989) e no artigo de Tanomaru (Tanomaru, 1995).

As principais etapas de um algoritmo genético são:

1) Codificação;

2) Geração da população inicial;

3) Cálculo da avaliação e aptidão;

4) Seleção dos indivíduos;

5) Cruzamento dos indivíduos;

6) Mutação;

8) Definição dos critérios de parada.

2.4.1 *Codificação:*

A definição de como codificar as soluções nos cromossomos é o primeiro passo para a construção de um algoritmo genético.

A representação por meio de cromossomos binários é a escolha tradicional para codificação das variáveis do algoritmo genético. Segundo Srinivas e Patnaik (1994), isso se deve a dois fatores: a) A simplificação da implementação, especialmente no tocante às operações genéticas, que devem ser redefinidas se a representação binária não for utilizada; b) Um resultado teórico derivado da teoria dos esquemas que diz que, quanto menor a cardinalidade do alfabeto utilizado na codificação, i.e., quanto menor o número de símbolos possíveis, maior será o número de esquemas processados simultaneamente e portanto mais eficiente será o algoritmo genético.

Na grande maioria das aplicações, as soluções são codificadas em um único cromossomo e, por isso, às vezes, refere-se indistintamente a indivíduos e cromossomos como se fossem a mesma entidade. Em grande parte dos algoritmos genéticos, os cromossomos são binários, i.e., constituídos por uma seqüência de 0s e 1s e portanto o esquema de codificação deve mapear o espaço de busca original em

seqüências binárias. Para variáveis binárias, a codificação é trivial: os valores de cada variável são simplesmente copiados nos cromossomos.

Para o problema em pauta, cada cromossomo será representado pelo conjunto de blocos de valores positivos contidos no modelo de blocos que serão lavrados considerando as restrições geotécnicas. Nesse caso, o cromossomo será do tamanho do número de blocos de minério que existe no modelo de blocos e corresponde a um indivíduo que é um candidato à solução do problema.

2.4.2 *População inicial*

Em geral, a população é inicializada com pontos escolhidos aleatoriamente mas, dependendo da aplicação, podem existir formas heurísticas de selecionar uma população inicial mais favorável.

É importante que a população inicial esteja bem distribuída pelo espaço de busca. Primeiro, para que todas as partes da população sejam representadas e, segundo, porque o algoritmo genético necessita de uma boa diversidade inicial de combinações para funcionar bem, da mesma forma que populações de organismos vivos precisam de uma boa diversidade genética para se adaptar eficientemente a seu meio ambiente.

A eliminação de duplicatas (Davis, 1991) é uma adaptação óbvia e interessante uma vez que a presença de indivíduos idênticos numa população implica em uma menor diversidade genética. Para se fazer isso, deve-se, primeiro, testar se existem indivíduos idênticos na população. De cada grupo de indivíduos idênticos encontrados, eliminam-se todos menos um e substituem-se os eliminados por novos indivíduos que podem ser gerados aleatoriamente. Esse método também demanda custo computacional extra, devido à necessidade de se comparar todos os indivíduos e de gerar substitutos.

2.4.3 *Avaliação e aptidão*

Cada indivíduo da população é avaliado no início de cada geração. Essa avaliação é feita externamente ao algoritmo genético. O resultado dela deve ser um valor numérico que expresse a qualidade relativa da solução codificada em cada indivíduo. A avaliação correlaciona-se portanto com a função objetivo do problema,

embora não seja necessário que exista uma função objetivo explicitamente formulada. Os valores provenientes da avaliação de cada indivíduo devem então se transformar em outro valor, necessariamente positivo, denominado aptidão, que será utilizado para definir as probabilidades de reprodução de cada um, na fase de seleção (Srinivas e Patnaik, 1994).

No problema em pauta, para cada cromossomo gerado, que será um indivíduo caracterizado por um conjunto de blocos de minério, a avaliação será o valor da cava (benefício do minério + despesas do estéril) considerando-se as restrições geotécnicas. A avaliação será realizada por uma função específica externa ao algoritmo genético que, em função dos blocos de minério do indivíduo, extrairá também os blocos de estéril considerando-se os ângulos dos taludes.

O valor de aptidão dos indivíduos também pode ser calculado de muitas maneiras. A forma mais simples e óbvia é utilizar-se o próprio valor da avaliação (função objetivo) para cada indivíduo o que é, às vezes, chamado de *aptidão bruta*. Entretanto, como a aptidão tem que ser necessariamente um valor positivo, para se estabelecerem as probabilidades de seleção dos indivíduos, pode ser impossível usar diretamente o valor da função objetivo, dependendo do problema. Mesmo quando é possível utilizar a aptidão bruta, a prática tem indicado que essa não é, normalmente, a melhor opção (Tanomaru, 1995), e diversas abordagens são propostas para transformar os valores da função objetivo de forma a melhorar o desempenho dos algoritmos genéticos.

A principal preocupação no cálculo da aptidão, é a pressão *seletiva*. A pressão seletiva pode entender-se como a intensidade com a qual o algoritmo genético favorece os indivíduos mais aptos em detrimento dos menos aptos, o que está basicamente relacionado às probabilidades de reprodução dos indivíduos. A pressão seletiva influencia diretamente a diversidade da população: uma alta pressão seletiva faz com que a diversidade diminua muito rapidamente, podendo ocasionar uma convergência prematura, enquanto uma baixa pressão seletiva preserva a diversidade mas dificulta a convergência para o ótimo. Ao se analisar a execução de um algoritmo genético que use a aptidão bruta, percebe-se que a pressão seletiva é usualmente mais alta no início, porque os indivíduos são bastante diferentes e, por isso, tendem a apresentar valores muito díspares para a função objetivo. No final da execução, ao contrário, a pressão seletiva é normalmente baixa, porque os indivíduos são mais

parecidos e tendem a apresentar valores bem semelhantes para a função objetivo. O que se deseja entretanto é justamente o oposto: uma pressão seletiva baixa, no início para que os indivíduos menos aptos tenham alguma chance de reprodução e, assim se possa explorar melhor o espaço de busca; e uma pressão seletiva mais alta, no final, porque como os indivíduos estão agora muito parecidos, fica mais difícil, para os métodos de seleção, privilegiar os mais aptos, podendo prejudicar a convergência do algoritmo genético.

O escalamento de aptidão é uma adaptação que visa regular a pressão seletiva, durante o andamento do algoritmo, tentando obter os efeitos desejados citados anteriormente. No escalamento, os valores da função objetivo, para cada indivíduo, são transformados de modo que as aptidões resultantes induzam um nível de pressão seletiva desejado durante a evolução do algoritmo genético. O tipo de escalamento mais utilizado é o *escalamento linear*, proposto por Goldberg (Goldberg, 1989).

A *janela de aptidão* é outra adaptação usada para converter uma função objetivo em uma função aptidão válida. Nela, a aptidão é calculada subtraindo-se do valor da função objetivo de cada indivíduo, o valor mínimo desta para toda a população, em cada geração. Esse método é muito simples de se implementar e garante que valores de aptidão negativos não sejam gerados. Mas não é tão eficaz na regulação da pressão seletiva, como é o escalamento linear.

No método de ordenação (Baker, 1985), não se usam os valores da função objetivo diretamente para o cálculo das aptidões. Ao invés disso, esses valores são utilizados para ordenar os indivíduos da população, a cada geração, e a aptidão é então atribuída a cada indivíduo, em função de sua posição nessa ordenação (*ranking)*. A forma mais simples de se atribuir um valor de aptidão utilizando um procedimento de *ranking* é ordenar os indivíduos do pior para o melhor e dar-lhes um valor de aptidão igual à posição de cada um no ordenamento. Se houver, por exemplo, uma população de 100 indivíduos, o pior receberá aptidão 1, o segundo pior aptidão 2 e assim por diante até o melhor, que terá aptidão 100. É fácil perceber que, dessa forma, a pressão seletiva mantem-se constante durante a execução do algoritmo e muitas implementações práticas utilizam esse método.

A última adaptação referente ao cálculo da aptidão é o chamado *compartilhamento* que foi inspirado no conceito de *nicho ecológico*. Duas espécies diferentes ocupam o mesmo nicho. Quando os recursos de que precisam para sobreviver (comida, abrigo, locais de reprodução, etc.) são muito parecidos, há intensa competição entre as espécies. Se, ao contrário, os nichos que as espécies ocupam são distintos, i.e. se os recursos dos quais as espécies necessitam são suficientemente diferentes, a competição entre elas será pequena ou inexistente. A extensão da idéia de nicho para o algoritmo genético simples considera que indivíduos que estão mais próximos no espaço de busca estariam ocupando nichos muito parecidos, enquanto os indivíduos mais afastados estariam em nichos bastante diferentes. Por isso, no compartilhamento, os valores de aptidão são modificados de forma que aqueles indivíduos que estão mais distantes dos outros tenham a aptidão aumentada, enquanto os indivíduos mais próximos no espaço de busca tenham seus valores de aptidão diminuídos.

O uso do compartilhamento diminui a pressão seletiva sobre os indivíduos mais distantes e aumenta a pressão sobre os mais próximos. Dessa forma, ele "incentiva o distanciamento" entre os indivíduos da população, o que aumenta a diversidade e provoca uma exploração mais global no espaço de busca. Embora o compartilhamento seja aparentemente vantajoso, ele acrescenta algum esforço computacional, por causa do cálculo das distâncias e tem mostrado, na prática, uma tendência a dificultar a convergência para o ótimo (Tanomaru, 1995). Experimentos com o uso do compartilhamento (Goldberg, 1989) mostram que, para funções objetivo multimodais, os algoritmos genéticos que não utilizam compartilhamento exibem um padrão onde os indivíduos se concentram ao redor de algum dos máximos enquanto que, nos algoritmos genéticos que utilizam o compartilhamento, os indivíduos concentram-se ao redor de alguns dos máximos presentes, sendo o número de indivíduos, ao redor de cada máximo, proporcional à altura relativa deles.

2.4.4 *Seleção*

Os algoritmos genéticos utilizam o modelo populacional geracional onde todos os indivíduos da população anterior são eliminados e substituídos a cada geração. Porém existem maneiras alternativas de compor as novas populações a cada geração. Na denominada reprodução de estado estável (Syswerda, 1991), produz-se, a cada geração, um número de descendentes, correspondente a apenas uma fração *N/r* da

população, que irá substituir um número equivalente de indivíduos da população anterior, em geral os N/r piores indivíduos. Esse método implica em algum trabalho computacional adicional, visto que os indivíduos da população devem ser ordenados para que se escolham os N/r piores. Mas tem a grande vantagem de permitir que se trabalhe com populações maiores, diminuindo-se a perda de variabilidade.

A adaptação de modelo populacional mais utilizada é o chamado elitismo. Nele, toda a população é, em primeira instância, substituída por seus descendentes, como nos algoritmos genéticos geracionais. Porém, se os melhores indivíduos da população anterior não estiverem presentes na nova população devido à seleção ou às operações genéticas, eles são acrescentados no lugar dos piores indivíduos “novos”. Em princípio, esse método também requereria a ordenação dos indivíduos para a escolha dos melhores mas isso, na verdade, não é necessário para o tipo mais comum de elitismo onde apenas o melhor indivíduo é preservado. Sua principal vantagem é garantir que a(s) melhores(s) soluções não sejam perdidas durante a execução do algoritmo genético, mitigando possíveis efeitos deletérios da seleção, recombinação e mutação.

Nos chamados algoritmos genéticos de geração contínua (Michalewicz, 1992), faz-se uma inversão na seqüência normal de operações do algoritmo genético e ocorre uma competição direta entre os indivíduos da população anterior e seus descendentes. Primeiro, todos os indivíduos são copiados. Essas cópias sofrem a recombinação e a mutação da forma usual, são avaliadas e recebem valores de aptidão. Depois se juntam os indivíduos originais da população e as cópias feitas no passo anterior, formando uma população temporária de 2*n* indivíduos. Sobre essa população temporária faz-se uma seleção baseada na aptidão dos 2*n* indivíduos, utilizando-se algum dos métodos de seleção já estabelecidos, de forma a se ter novamente uma população de *n* indivíduos, Todo esse processo completa uma geração do algoritmo genético e é repetido na execução das próximas gerações.

Outra adaptação de modelo populacional é o chamado genocídio periódico. Assim como a reprodução de estado estável, o genocídio periódico visa combater a perda de variabilidade que pode ser crítica quando se trabalha com populações pequenas. A idéia básica é executar um algoritmo genético geracional que, em intervalos de um certo número de gerações, sofra uma intervenção. Nessa intervenção, a população é aumentada drasticamente, para 10*n* por exemplo e,

depois, passa por uma seleção que reduz o número a *n* novamente, de forma semelhante ao que se faz nos algoritmos genéticos de geração contínua. Os indivíduos adicionais podem vir de um processo reprodutivo alterado onde se fazem muito mais cópias, ou podem ser gerados aleatoriamente. Esse método apresenta um custo computacional significativamente maior, devido à necessidade de se avaliar em muitos indivíduos em certas gerações e, além disso, ainda introduz novos parâmetros que devem ser definidos, como a freqüência do genocídio e o quanto a população deve aumentar antes dele.

No modelo populacional conhecido como superseleção, divide-se a população em castas, de acordo com os valores de aptidão e realiza-se uma reprodução estratificada, com os indivíduos das castas com maior aptidão reproduzindo-se mais, em média, do que os das castas de menor aptidão. Por exemplo, pode-se dividir uma população de 200 indivíduos em 3 castas: os melhores 10 indivíduos, os próximos 40 e os outros 150. Pode-se, então, estipular, por exemplo, que os 10 melhores indivíduos deixem 50 descendentes, que os 40 indivíduos da segunda casta tenham 80 descendentes e os 150 da casta mais baixa produzam apenas 70 descendentes para a próxima geração. Esse método visa contornar os problemas da seleção via método da roleta, diminuindo a chance de que indivíduos com alta aptidão não deixem descendentes. Esse procedimento acarreta pequeno custo computacional adicional por causa da necessidade de se ordenarem os indivíduos e de se fazer seleção em separado para cada casta.

Nesse modelo, deve-se primeiro verificar se existem indivíduos idênticos na população após a seleção, a reprodução e as operações genéticas. De cada grupo de indivíduos idênticos encontrados, eliminam-se todos menos um e substituem-se os eliminados por novos indivíduos, que podem ser gerados aleatoriamente ou obtidos por seleção e operações genéticas. Esse método também apresenta um custo computacional extra, devido à necessidade de se compararem todos os indivíduos e de se gerarem substitutos, caso indivíduos idênticos sejam encontrados.

No processo de seleção, decidem-se quais os indivíduos da população se reproduzirão e quantos descendentes cada um deixará para a próxima geração. Nesse estágio do algoritmo genético, os descendentes são apenas cópias dos genótipos dos indivíduos que foram selecionados mas serão posteriormente alterados pelas operações genéticas. No algoritmo genético, a definição de quais indivíduos

serão copiados e de quantas cópias cada um deixará é feita, a partir do valor de aptidão de cada indivíduo, que é usado para definir as probabilidades de reprodução deles.

No algoritmo genético simples, usa-se um esquema de *reprodução geracional* onde se gera um número de descendentes igual ao número de indivíduos da população e substitui-se toda a população por esses descendentes, a cada geração, mantendo-se o tamanho da população constante. Se forem produzidas *n* cópias a cada geração e *Pi* é a probabilidade de que uma determinada cópia seja descendente do i-ésimo indivíduo, então se pode concluir que o número esperado de descendentes para esse indivíduo, é *nPi*. Assim, indivíduos com valores de aptidão acima da média tenderão a deixar mais de uma cópia, enquanto aqueles com aptidão abaixo da média tenderão a deixar menos de uma cópia. O número esperado de cópias, para cada indivíduo, *nPi*, é quase sempre fracionário, mas o número de cópias efetivamente deixadas, por cada indivíduo, tem que ser forçosamente um inteiro. Esse problema é solucionado no algoritmo genético utilizando-se o *esquema de seleção da roleta*. Nesse esquema, pensa-se em uma roleta onde cada casa refere-se a um dos indivíduos. Ao contrário de uma roleta de cassino onde todas as casas têm o mesmo tamanho, nessa roleta fictícia, os tamanhos das casas são diferentes e proporcionais à probabilidade de seleção de cada indivíduo. A seleção dos indivíduos para reprodução é feita então “girando” a roleta *n* vezes e copiando o indivíduo que foi sorteado em cada rodada.

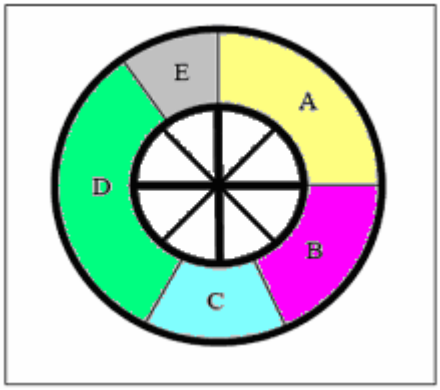


Figura 2.4.4.1 – Representação do esquema de seleção da roleta.

Na figura anterior, cada um dos indivíduos (A, B, C, D e E) tem uma casa na roleta cujo tamanho é proporcional a sua aptidão. Cada vez que a roleta é girada, um

dos indivíduos é selecionado para ser copiado e, posteriormente, sofrer as operações genéticas.

É importante notar que esse processo é aleatório e os indivíduos com valores de aptidão mais alta não necessariamente serão copiados, embora tenham chance proporcionalmente maior de serem escolhidos. Da mesma forma, esse processo tampouco impede que os indivíduos com aptidões mais baixas sejam selecionados, embora tenham menos chances. Assim, o método da roleta introduz um certo grau de aleatoriedade no processo, permitindo que o número efetivo de cópias deixadas para cada indivíduo varie consideravelmente em relação ao número esperado.

O método da roleta é um bom exemplo para o processo de seleção que ocorre nos organismos vivos pois os indivíduos mais aptos têm mais chance mas não têm a garantia de deixar descendentes. Entretanto essa aleatoriedade introduzida pelo método da roleta faz com que muitos indivíduos não se reproduzam com taxas semelhantes a seu número esperado de cópias, o que dificulta a convergência do algoritmo genético a longo prazo. Por isso, foram desenvolvidas algumas adaptações para o processo de seleção, que substituem o método da roleta na tentativa de se contornar esse problema. A seguir, faz-se uma descrição de algumas dessas adaptações.

O método do *valor esperado* ou *amostragem estocástica sem reposição*, usa o mesmo princípio da roleta mas limita o número de cópias que cada indivíduo pode deixar em função de sua probabilidade de seleção. A diferença básica com relação ao método da roleta, é que, da seleção, calcula-se o número esperado de cópias para todos os indivíduos e, para cada vez que um indivíduo é selecionado, subtrai-se 1 de seu número esperado de cópias. Quando o número esperado de cópias de qualquer indivíduo ficar nulo ou negativo, ele fica impedido de ser sorteado novamente. Assim, o número de cópias de qualquer indivíduo nunca ultrapassa o número esperado de cópias, arredondado para o próximo inteiro.

O método de *amostragem determinística* também se baseia no número esperado de cópias para cada indivíduo. Só que aqui se aloca imediatamente um número de cópias para cada indivíduo, correspondente à parte inteira de seu número esperado de cópias. Isso define o primeiro grupo de m cópias onde $m \leq n$ (o tamanho da população). Depois, os indivíduos são ordenados, de acordo com a parte

fracionária de seu número esperado de cópias, de forma decrescente. As *n* – *m* cópias que ainda faltam para completar a população são atribuídas aos *n-m* primeiros indivíduos dessa lista ordenada.

Assim como na amostragem determinística, no método do *resto estocástico*, também se aloca um número de cópias de indivíduos correspondente à parte inteira dos números esperados de cópias de cada um. A diferença entre os dois métodos é que, no resto estocástico, as partes fracionárias são usadas para estabelecer novas probabilidades de seleção, a partir das quais as *n-m* cópias restantes são alocadas por sorteio, de forma idêntica ao método da roleta.

Dentre os métodos alternativos de seleção, o *torneio estocástico* é o mais singular. Nele se utiliza o método da roleta para sortear *n* grupos com um pequeno número de indivíduos em cada. De cada grupo, copia-se aquele indivíduo com o valor mais alto de aptidão, obtendo-se *n* cópias no final do processo.

Todos esses métodos resultam em um processo de seleção menos exposto a flutuações aleatórias do que o método da roleta mas não existe consenso sobre se eles são melhores e em quais situações (Tanomaru, 1995).

2.4.5 *Recombinação*

As *n* cópias geradas no processo de seleção formam o chamado *mating pool*, sobre o qual serão efetuadas as operações de recombinação e mutação. Para a recombinação, primeiro os cromossomos do *mating pool* são agrupados em duplas. Depois, para cada dupla, escolhe-se aleatoriamente um número, com distribuição uniforme entre 0 e 1. Nas duplas onde o número sorteado for menor ou igual a um dado valor, chamado probabilidade de recombinação (*Prec*), a recombinação será efetuada. No algoritmo genético, a recombinação é feita pelo chamado cruzamento (*crossover)* de um ponto onde se escolhe, para cada dupla de cromossomos que serão efetivamente recombinados, *um ponto de corte* a partir do qual permutam-se, entre dois cromossomos da dupla, todos os valores correspondentes às disposições posteriores, como ilustrado na figura 2.11. Para se escolher o ponto de corte simplesmente sorteia-se, para cada dupla de cromossomos que sofrerá recombinação, um número com distribuição uniforme entre 1 e *c*-1, onde *c* é o comprimento do cromossomo. Serão permutadas então, as posições cujos índices nos cromossomos sejam maiores que o número sorteado.

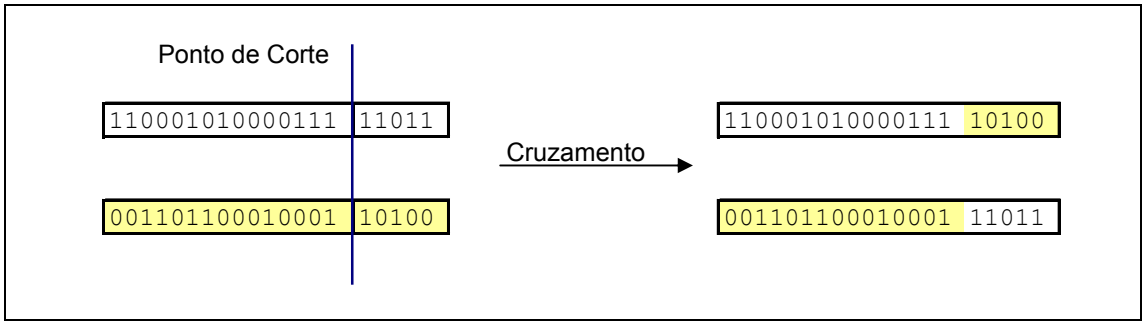


Figura 2.4.5.1 – Cruzamento com um ponto.

Devido ao importante papel que a recombinação desempenha nos algoritmos genéticos, há vários estudos sobre seu funcionamento e suas características. Muitas adaptações e operadores alternativos são propostos e analisados. A maioria foi feita para adaptar os operadores a representações cromossômicas diferentes, usadas em problemas específicos. Essas adaptações têm o mérito de ilustrar como é possível melhorar o desempenho de um algoritmo genético desenvolvendo operadores sob medida. Alguns operadores de recombinação alternativos têm um caráter genérico, podendo ser aplicados a algoritmos genéticos em geral.

No cruzamento com dois pontos, escolhem-se dois pontos de corte, e trocam-se, entre os cromossomos que formam pares, as seqüências que se encontrarem entre esses dois pontos de corte. Esses pontos de corte devem ser escolhidos sorteando-se dois números entre 0 e *c* (onde *c* é o comprimento do cromossomo), e não entre 1 e *c*-1 como no *crossover* de um ponto, para permitir também a troca dos bits das extremidades. Eventualmente ambos os pontos de corte podem cair na mesma posição. Nesse caso, pode-se fazer outro sorteio ou simplesmente não se trocar posição nenhuma. Também é possível generalizar a idéia do *crossover* de dois pontos estabelecendo operadores de crossover de três ou mais pontos, dependendo do tamanho dos cromossomos. Tais adaptações entretanto não são relevantes do ponto de vista teórico e são muito utilizadas.

Outra adaptação importante do operador de recombinação é o chamado c*ruzamento* uniforme, proposto por Syswerda (1989). Nesse método, gera-se, para cada par de recombinações formado, uma seqüência binária aleatória com o mesmo comprimento dos cromossomos, que se denomina padrão de recombinação. Para cada par de cromossomos, percorre-se o padrão de recombinação correspondente e,

toda a vez que se encontra um “1” nele, trocam-se os bits que estejam na posição equivalente. Obviamente, sempre que se encontra um “0” no padrão, não se trocam os símbolos equivalentes.

Analisando-se o funcionamento dos operadores de recombinação já descritos, conclui-se que os cruzamentos de um e dois pontos tendem a preservar mais os esquemas, enquanto o cruzamento uniforme é mais disruptivo. Em compensação, o cruzamento uniforme tende a promover uma exploração mais global do espaço de busca do que os outros dois. Srinivas e Patnaik (1994) sugerem que o *crossover* uniforme é mais adequado para populações pequenas, porque mantém a diversidade e aumenta a exploração do espaço de busca. Por outro lado, para populações grandes onde a diversidade é fatalmente maior, pode ser mais vantajoso utilizar-se o cruzamento de um ou dois pontos pois preservam mais os blocos construtores, o que pode acelerar a convergência.

Além dos próprios operadores de recombinação, pode-se também modificar a forma pela qual os pares são escolhidos. Normalmente os pares para a recombinação são formados sorteando-se, sem reposição, dois indivíduos no *mating pool* de cada vez. Ao invés disso, pode-se usar algum critério para escolher os pares diretamente, ou influenciar em seu sorteio. Uma idéia já testada é formar duplas de indivíduos de alta aptidão com outros também de alta aptidão e, correspondentemente, os de baixa aptidão com outros de baixa aptidão. Essa abordagem pode aumentar a velocidade de convergência mas também pode prejudicar a exploração global do espaço de busca.

2.4.6 *Mutação*

Para se realizar a mutação, sorteia-se para cada posição de cada cromossomo, um número com distribuição uniforme entre 0 e 1, e compara-se com a *Pmut* (probabilidade de mutação). Onde o número sorteado for menor ou igual a *Pmut* troca-se o símbolo (alelo) que ocupa a posição correspondente. Como o algoritmo genético usa sempre cromossomos binários, as únicas trocas possíveis são de 1 para 0 e de 0 para 1.

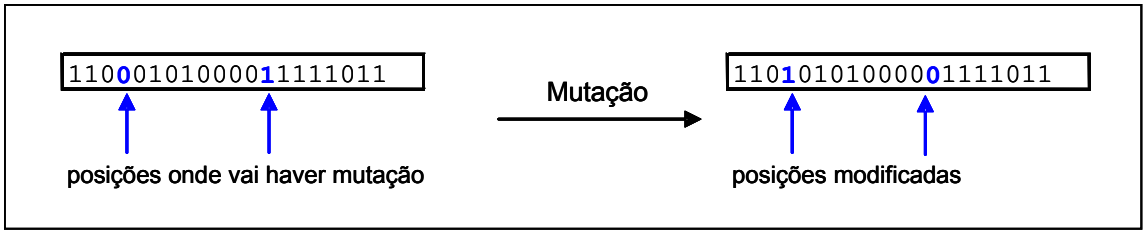


Figura 2.4.6.1 – Mutação na codificação binária.

Note-se que as mutações no algoritmo genético são independentes, i.e., o fato de ocorrer ou não uma mutação, em uma dada posição, não altera a probabilidade de ocorrerem mutações nas outras posições.

Existem ainda métodos que alteram as probabilidades de cruzamento e mutação para cada indivíduo, de acordo com seu desempenho e com o desempenho médio e máximo encontrados na população. O objetivo é manter um nível adequado de diversidade genética no âmbito da população e evitar a convergência prematura. Se o número de indivíduos da população é pequeno e/ou o algoritmo já executou inúmeras iterações, provavelmente há pouca diversidade genética dentro da população. Se não houver um mecanismo que restaure essa diversidade, pode-se comprometer a convergência para o ótimo global (Vasconcelos, 2004). Para a aplicação desse método, utilizam-se as equações a seguir:

$$P_c = k_1 \frac{(f_{\max} - f')}{(f_{\max} - f_{med})} \text{ se } f' \geq f_{med} \qquad (2.4.6.1)$$

$$P_c = k_3 \text{ se } f' < f_{med} \qquad (2.4.6.2)$$

$$P_m = k_2 \frac{(f_{\max} - f)}{(f_{\max} - f_{med})} \text{ se } f \geq f_{med} \qquad (2.4.6.3)$$

$$P_m = k_4 \text{ se } f < f_{med} \qquad (2.4.6.4)$$

onde:

P_c: probabilidade de cruzamento;

P_m: probabilidade de mutação;

f': representa o maior valor da função desempenho entre os dois membros participantes do cruzamento;

f : valor da função avaliada para o indivíduo sujeito à mutação;

$f_{\max}$: valor de desempenho máximo;

f_{med} : valor de desempenho médio;

k_1, k_2, k_3 e k_4 : usualmente $k_1=k_3=1$ e $k_2=k_4=0,05$.

2.4.7 *Penalidades e restrições*

A questão da limitação do espaço de busca a um determinado intervalo é um ponto-chave quando existem restrições sobre as variáveis de decisão da função objetivo. Para que uma determinada combinação dessas variáveis tenha validade, deverão atender a todos os critérios de busca do domínio da função.

Uma maneira de se contornar esse tipo de problema seria executar a avaliação de cada indivíduo da população corrente: caso um determinado integrante não atenda as restrições, procura-se outro para satisfazê-las. Esse tipo de análise, usualmente, não é eficaz. Outra opção seria acoplar as restrições sobre as variáveis à função objetivo, método conhecido na literatura como *método das penalidades*.

Nos métodos que lidam com penalidades, um problema com restrições pode ser transformado em um problema irrestrito associando uma função de penalidade à função objetivo. Dessa forma, um problema de maximização com restrição pode ser escrito matematicamente como sendo:

$$\text{Max } f(x) \qquad (2.4.7.1)$$

Sujeito a "*m*" restrições de igualdade e "*l*" restrições de desigualdade:

$$g_j(x) \leq 0 \quad \text{com} \quad j = 1,2,3,...,m \qquad (2.4.7.2)$$

$h_k(x) = 0$ com $k = 1,2,3,...,l$ (2.4.7.3)

onde as funções f(x), $g_j(x)$ e $h_k(x)$ são respectivamente a função objetivo, j-ésima restrição de desigualdade e k-ésima restrição de igualdade.

Assim, o problema pode ser reescrito na forma de:

$$\text{Max } \theta(x,r,s) = f(x) + r\sum_{j=1}^{m}\left\{\max\left[0, g_j(x)\right]\right\}^2 + s\sum_{k=1}^{l}\left[h_k(x)\right]^2 \quad (2.4.7.4)$$

onde as constantes *r* e *s* correspondem aos pesos de cada um dos tipos de restrições violadas.

Existem, ainda, métodos que penalizam dinamicamente, ou seja, na função a ser otimizada, pode-se incluir um parâmetro de acordo com um contador de geração *t* (Joines e Houck, 1994):

$$\text{Max } \theta(x,r,s) = f(x) + \left(C.t\right)^{\alpha}\left(r\sum_{j=1}^{m}\left\{\max\left[0, g_j(x)\right]\right\}^{\beta} + s\sum_{k=1}^{l}\left[h_k(x)\right]^{\beta}\right) \quad (2.4.7.5)$$

onde *C* e *α* são constantes definidas pelo usuário e *β* geralmente possui valor igual a 2.

2.4.8 *Definição dos parâmetros*

A probabilidade de recombinação, a probabilidade de mutação e o tamanho da população são definidos antes da execução de um algoritmo genético. Infelizmente, não existe uma metodologia que possa definir os melhores valores para esses parâmetros em relação a um problema específico. Por isso, é necessário testar diversas possibilidades e investigar qual delas apresenta os melhores resultados. Normalmente se empregam em algoritmos genéticos, valores nas seguintes faixas: *n*: 10 a 200, *Prec*: 0,5 a 1,0 e *Pmut*: 0,001 a 0,10.

Nos algoritmos genéticos em sua forma tradicional, os parâmetros de controle, como *Pmut* e *Prec*, são definidos pelo usuário no início do algoritmo e permanecem com os mesmos valores até o final. Entretanto, como a população muda de acordo com a evolução do algoritmo genético, é razoável pensar-se que parâmetros

adequados no início do algoritmo genético possam não ser muito eficientes mais à frente, em sua evolução. Nos chamados algoritmos genéticos adaptativos, alteraram-se os valores dos parâmetros de forma a aumentar a sua eficiência. O tipo mais comum de algoritmo genético adaptativo é aquele no qual varia-se *Pmut* de forma a regular o balanço "exploração/explotação". À medida que o algoritmo genético vai evoluindo, os indivíduos tendem a ficar mais parecidos e, assim, a recombinação vai perdendo a capacidade de explorar o espaço de busca. Porém, se se aumenta a taxa de mutação, pode-se criar mais diversidade na população, explorando-se uma área maior do espaço de busca. Para se definir quando se deve aumentar *Pmut*, monitoram-se as distâncias entre os indivíduos da população, que refletem a homogeneidade desta. Caso seja necessário, aumenta-se a taxa de mutação de forma a recuperar a diversidade e manter o poder de busca global.

Curiosamente em outra abordagem de algoritmo genético adaptativo, adota-se um procedimento de certa forma contrário ao anterior. Sabe-se que altas taxas de mutação tornam a busca do algoritmo genético mais global, mas também dificultam a sua convergência. Assim, um procedimento razoável para variar a taxa de mutação é começar com um *Pmut* alto, de forma a explorar amplas regiões do espaço de busca no início e diminuí-lo paulatinamente, à medida que o algoritmo genético evolua e o efeito disruptivo da mutação se torne um empecilho para sua convergência. Ambas as abordagens fazem sentido em relação à teoria dos algoritmos genéticos e não está claro ainda qual delas seria melhor. Provavelmente a eficácia de cada uma está relacionada a outros fatores, como o tamanho da população, a natureza da função objetivo e a própria evolução que o algoritmo genético está apresentando (Srinivas e Patnaik, 1994).

2.4.9 *Critérios de Parada*

O ciclo de gerações deve ser repetido até que um certo critério de parada seja alcançado. O ideal seria prosseguir com o algoritmo até se encontrar o máximo ou mínimo da função. Mas isso só é possível em problema-exemplo onde já se conhece a solução e se quer apenas provar que tal método permite de encontrá-la. Em problemas práticos, não se consegue conhecer todo o espaço de busca. Portanto não se é capaz de reconhecer o ótimo global, mesmo que pudesse encontrá-lo.

Alternativamente se pode parar o algoritmo genético quando uma solução satisfatória for encontrada. Isso pode funcionar bem para alguns tipos de problemas mas, em muitos casos, pode ser difícil determinar o que é uma solução satisfatória ou então o próprio conceito de solução satisfatória pode não ter sentido pois se deseja encontrar a melhor solução possível.

Os dois critérios de parada mais simples e mais freqüentemente utilizados são: fixar um número de gerações ou fixar um tempo máximo de processamento. Outro critério, teoricamente melhor, seria parar o algoritmo quando ele chegasse à *estagnação*. Para isso, seria necessário definir, de forma objetiva, como se poderia reconhecer que o algoritmo genético chegou à estagnação. Mas isso pode ser muito complicado porque freqüentemente a evolução de um algoritmo genético ocorre de forma errática, com a aptidão média da população podendo oscilar fortemente ou apresentar longos períodos de aparente estagnação antes de conseguir alcançar melhorias significativas.

2.5 Otimização multiobjetivo utilizando algoritmos genéticos

Das diferenças entre os algoritmos genéticos e os métodos de otimização tradicionais, citam-se (Goldberg, 1989):

i) Os algoritmos genéticos trabalham com uma população de soluções em cada iteração ao invés de uma única solução. Em cada iteração, os algoritmos genéticos processam um conjunto de soluções. Essa característica é denominada paralelismo implícito;

ii) Os algoritmos genéticos não precisam de informação adicional a não ser o valor de aptidão das soluções. Isso faz com que os algoritmos genéticos sejam aplicáveis a uma grande variedade de problemas, para alguns dos quais não têm informações *a priori;*

iii) Os algoritmos genéticos empregam regras probabilísticas para guiar a busca. Por exemplo, o processo de seleção é baseado na aleatoriedade de duas soluções (seleção por torneio), ou na probabilidade de escolha (seleção proporcional) dessas soluções. O

operador de mutação permite que os algoritmos genéticos não recaiam nos ótimos locais, mudando a busca para outra região no espaço. Além de que, as soluções da população inicial podem ser escolhidas arbitrariamente. Em contrapartida, uma técnica de regras fixas não terá como escapar de ótimos locais, em caso de má decisão sobre a direção de busca.

A tabela a seguir exibe uma lista dos principais modelos de algoritmos genéticos aplicados a problemas multiobjetivos, com seus autores conforme (Deb, 2001).

Sigla	Nome do modelo	Autores
VEGA	Vector Evaluated Genetic Algorithm	[Schaffer, 1985]
WBGA	Weight Based Genetic Algorithm	[Hajela e Lin, 1992]
MOGA	Multiple Objective Genetic Algorithm	[Fonseca e Fleming, 1993]
NSGA	Non-Dominated Sorting Genetic Algorithm	[Srinivas e Deb, 1994]
NPGA	Niched-Pareto Genetic Algorithm	[Horn et al., 1994]
PPES	Predator-Prey Evolution Strategy	[Laumanns et al., 1998]
REMOEA	Rudolph's Elitist Multi-Objective Evolutionary Algorithm	[Rudolph, 2001]
NSGA-II	Elitist Non-Dominated Sorting Genetic Algorithm	[Deb et al., 2000]
SPEA, SPEA2	Strenght Pareto Evolutionary Algorithm 1 e 2	[Zitzler e Thiele, 1998, Zitzler et al., 2001]
TGA	Thermodynamical Genetic Algorithm	[Kita et al., 1996]
PAES	Pareto-Archived Evoltionary Strategy	[Knowles e Corne, 1999]
MOMGA-I, MOMGA-II	Multi-Objective Messy Genetic Algorithm	[Veldhuizen, 1999]
Micro-GA	Multi-Objective Micro-Genetic Algorithm	[Coello Coello e Toscano Pulido, 2001]
PESA-I, PESA-II	Pareto Envelope-Base Selection Algorithm	[Corne et al., 2000, 2001]

Tabela 2.5.1 – Algoritmos genéticos multiobjetivos.

2.5.1 *Descrição do Algoritmo NSGA*

Tendo em vista que os melhores resultados obtidos por Dias (2000), em sua dissertação de mestrado “Algoritmos Genéticos Aplicados a Problemas com Múltiplos Objetivos” deveram-se à utilização do algoritmo baseado em ordenação não-dominada (*Nondominated Sorting Genetic Algorithm* - NSGA), a seguir tem-se uma breve descrição do mesmo, que será utilizado na resolução dos problemas propostos neste trabalho.

A partir de uma idéia de Goldberg (989), Deb e Srinivas (1993) propuseram um esquema de ordenação de soluções candidatas a serem soluções eficientes. O princípio básico consiste de um método de seleção por classificação buscando “bons” pontos candidatos. A técnica de partilha é empregada como ferramenta auxiliar para manter a diversidade da população sobre a fronteira Pareto-Ótimo.

Uma vez que o algoritmo é baseado em um procedimento de ordenação dos pontos candidatos a serem pontos eficientes da população, esse método é conhecido na literatura técnica como *Nondominated Sorting Genetic Algorithm* (NSGA).

O NSGA difere do Algoritmo Genético Simples (AGS) apenas na maneira como o método de seleção trabalha, uma vez que os operadores de cruzamento e mutação permanecem idênticos. Antes de o método de seleção atuar, a população é classificada com base na definição de pontos eficientes (conceito de Pareto-Ótimo). Os indivíduos eficientes presentes na população corrente são primeiramente identificados. Assim, esses indivíduos definirão a primeira fronteira eficiente e terão como valor, na função aptidão, um número não nulo. O mesmo valor da função de aptidão é assumido para todos da primeira fronteira para se ter uma única probabilidade de reprodução. Para se manter a diversidade da população, os indivíduos são submetidos à atuação da técnica de formação de nichos com função de partilha triangular sobre os seus valores de aptidão.

Com a função de partilha, obtem-se a função de aptidão degradada, a qual é definida como sendo a função de aptidão original do indivíduo dividida pela quantidade proporcional de indivíduos ao redor do mesmo (dado pelo tamanho do nicho). Após a partilha, os indivíduos eficientes são ignorados temporariamente para se processar o resto da população de modo a identificar indivíduos para a segunda fronteira eficiente.

Esses pontos eficientes assumirão um novo valor da função de aptidão que deverá ser menor que o valor mínimo da função de aptidão da fronteira anterior. O processo continua até que toda a população seja classificada em fronteiras eficientes ou não-dominadas.

A população é então reproduzida de acordo com a sua função de aptidão. Inicialmente foi proposto um operador de seleção do tipo SRS (*Stochastic Reminder Sampling*), mas qualquer tipo de seleção pode ser aplicado. Uma vez que os indivíduos da primeira fronteira apresentam o maior valor de aptidão, terão sempre mais cópias que o resto da população corrente. Dessa forma, leva-se a pesquisa para regiões do espaço viável nas proximidades da fronteira Pareto-Ótimo. Esse método apresenta uma convergência acelerada e o processo de partilha auxilia a distribuição dos indivíduos sob a fronteira eficiente.

Conforme pode ser observado no fluxograma do NSGA (figura 2.5.1.1), o algoritmo é similar ao algoritmo genético simples, com exceção da classificação das fronteiras eficientes e da operação de partilha. A partilha, em cada fronteira, é determinada pelo cálculo do valor da função de partilha $Sh(d_{ij})$ entre dois indivíduos da mesma fronteira como abaixo:

$$Sh(d_{ij}) = \begin{cases} 1 - (d_{ij}/\beta_{shared})^{\alpha} & \text{se } d_{ij} < \beta_{shared} \\ 0 & \text{se } d_{ij} \geq \beta_{shared} \end{cases} \qquad (2.5.1.1)$$

Nessa equação, o parâmetro d_{ij} corresponde à distância no espaço em estudo entre dois indivíduos *i* e *j* da fronteira atual. A variável β_{shared} corresponde à distância máxima, nesse espaço, permitida entre dois indivíduos quaisquer para se tornarem membros do nicho. O parâmetro α (geralmente com valor 2) tem por objetivo penalizar os indivíduos mais próximos. Algumas idéias sobre como definir esses parâmetros podem ser encontradas em Goldberg (991), Deb (1999) e Srinivas e Deb (1994). O parâmetro tamanho de nicho, N_{count}, é calculado somando-se os valores da função de partilha de todos os indivíduos presentes na fronteira eficiente. Finalmente, a função de partilha de cada indivíduo é calculada dividindo-se a função de aptidão definida na classificação pelo tamanho do nicho.

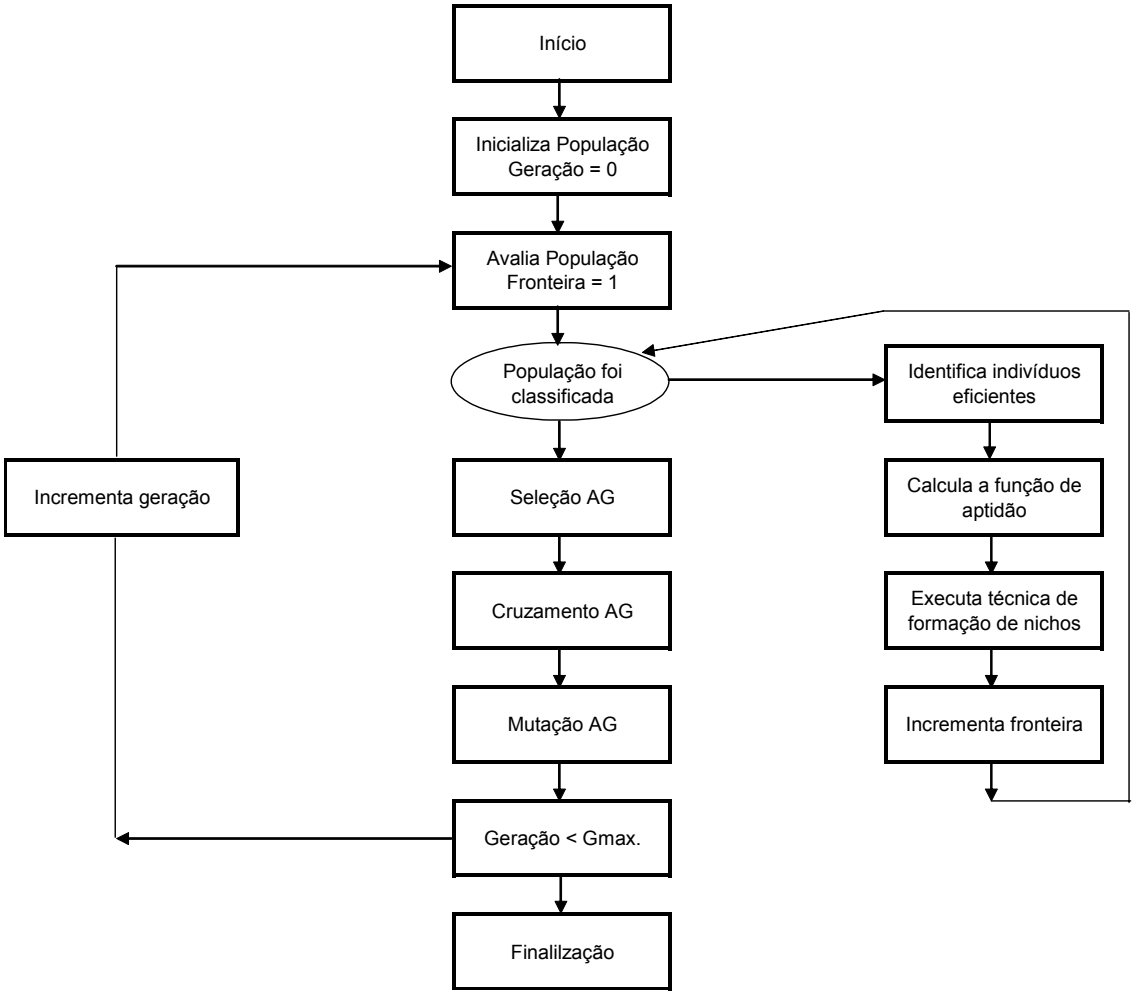


Figura 2.5.1.1 – Fluxograma NSGA (Dias, 2000).

Usualmente esse tipo de função de partilha é conhecida como partilha fenotípica (*phenotypic sharing*) (Deb, 1999). Em Goldberg (1987), Goldberg e Richardson sugeriram, mas não simularam, o uso da partilha baseada na proximidade genética. A proximidade genética de dois indivíduos pode ser tomada como o número de genes diferentes em seus cromossomos, ou seja, a distância *Hamming* entre as *strings* que os representam.

A distância d_{ij} considerada na função de partilha, baseada no espaço de variáveis de otimização, é a distância das seqüências de caracteres (*strings*) que identificam o indivíduo (considerando-se o caso de codificação binária) segundo Deb (1999) e Srinivas e Deb (1993). Para funções de apenas uma variável, essa distância pode ser calculada como a diferença absoluta da variável decodificada. No caso mais geral, para funções de *p*-variáveis, a distância pode ser calculada usando-se algum

critério de norma para espaços de dimensão *p*. Como critério de simplificação, a distância euclidiana é adotada na formulação da distância.

Dessa forma, dados os indivíduos com *p*-variáveis:

$$X_i = [X_{1,i} \quad X_{2,i} \quad X_{3,i} \quad ... \quad X_{p,i}] \tag{2.5.1.2}$$

$$X_j = [X_{1,j} \quad X_{2,j} \quad X_{3,j} \quad ... \quad X_{p,j}]$$

Onde X é a *k*-ésima variável decodificada do *i*-ésimo indivíduo.

A distância d_{ij} pode ser calculada como:

$$d_{ij} = \sqrt{\sum_{k=1}^{p} \left(X_{k,i} - X_{k,j} \right)^2} \tag{2.5.1.3}$$

Onde as variáveis $X_{1,i}$, $X_{2,i}$, ..., $X_{p,i}$ são parâmetros de otimização decodificados do problema em estudo.

Deb (1998), propôs outra fórmula de análise que considera a distância euclidiana para *p*-dimensões normalizada:

$$d_{ij} = \sqrt{\sum_{k=1}^{p} \left(\frac{X_{k,i} - X_{k,j}}{X\max_k - X\min_k} \right)^2} \tag{2.5.1.4}$$

onde são citadas duas estimativas de parâmetro β_{shared}:

$$[1] \qquad \beta_{shared} = \frac{1}{2}\left(\frac{\sqrt{p}}{\sqrt[p]{q}} \right) \tag{2.5.1.5}$$

$$[2] \qquad \beta_{shared} = \frac{1}{2}\sqrt[p]{0.1} \tag{2.5.1.6}$$

Outra maneira de se implementar a função de partilha é utilizar o espaço de funções objetivo ao invés do espaço de otimização. É sabido que diferentes escalas, nas dimensões do espaço de funções objetivo em um problema multiobjetivo, podem afetar a distribuição dos indivíduos na população corrente.

Dessa forma, sugere-se uma mudança de escala normalizando todos os valores das funções objetivo no intervalo [0,1]. Note-se que esse processo somente é possível quando se conhecem os valores máximo e mínimo que cada função objetivo pode assumir, o que é usualmente o caso.

Define-se então um valor sugestivo para β_{shared} como sendo:

$$\beta_{shared} = 0.5 * \sqrt[n]{\frac{1}{10}} \tag{2.5.1.7}$$

a qual é análoga à equação da opção [2] (equação 2.5.1.6) anteriormente citada. Para o cálculo sobre os valores das funções objetivo, deve-se inicialmente definir pesos positivos w_i para cada uma das funções, de modo que:

$$\sum_{i=1}^{n} w_i = 1 \tag{2.5.1.8}$$

A distância entre dois indivíduos d_{ij} pode então ser dada de forma análoga à distância no espaço de variáveis de otimização:

$$d_{ii} = \sqrt{\sum_{k=1}^{r} \left(\frac{F_{k,i} - F_{k,j}}{X\max_k - X\min_k} \right)^2} \tag{2.5.1.9}$$

3 Metodologia

3.1 Processo de Modelagem

Conforme descrito por Goldbarg e Luna (2005), é possível, de forma bastante geral, resumir o processo de modelagem ou de construção de modelos na ótica operacional, pelos passos sugeridos no fluxograma a seguir, que foi utilizado neste trabalho:

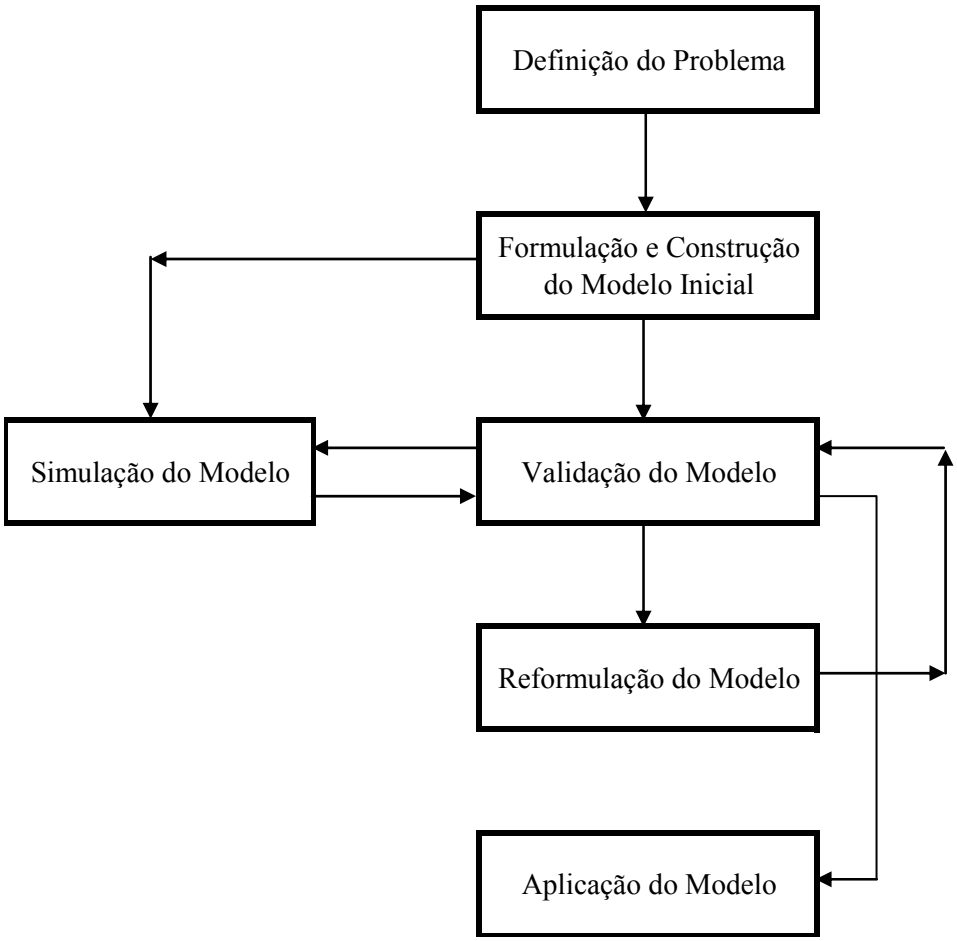


Figura 3.1.1 – Fluxograma do processo de modelagem (Goldbarg e Luna, 2005).

3.2 Definição dos problemas

O objetivo principal deste trabalho é avaliar a aplicabilidade dos algoritmos genéticos na definição de cava final e seqüenciamento de lavra para minas a céu aberto.

Para se atingir tal objetivo pretende-se desenvolver um conjunto de rotinas computacionais que, a partir de um modelo de blocos tecnológicos conhecido que represente uma jazida, seja capaz de gerar um conjunto de soluções ótimas, tendo em vista os objetivos de maximização de reservas (massa), maximização de valor não descontado (lucro), respeitando as condicionantes de qualidade e distribuição das massas no tempo (seqüenciamento de lavra).

Pode-se dividir nos seis problemas citados a seguir:

a) Otimização mono-objetivo
    I. Determinação da cava final maximizando o lucro não descontado;
    II. Determinação da cava final maximizando o lucro não descontado com restrições de qualidade;
    III. Determinação da cava final maximizando o lucro não descontado com restrições de qualidade e/ou massas no tempo.
b) Otimização multiobjetivo
    I. Geração da família de cavas ótimas visando maximizar recursos e o lucro não descontado;
    II. Geração da família de cavas ótimas visando maximizar recursos e teor médio;
    III. Geração da família de cavas ótimas visando maximizar o lucro não descontado e teor médio.

3.3 Construção dos modelos de blocos teóricos

O modelo de blocos utilizado neste trabalho é baseado no exemplo originalmente apresentado em Lerchs e Grossmann (1965), elaborado por Sainsbury (1970) e reapresentado por Hustrulid & Kuchta (1995), que descreve o algoritmo de Lerchs-Grossmann em 2D, conforme item 2.1.1.

Além dos valores econômicos dos blocos, são atribuídos valores de massa visando a geração de cavas multiobjetivo, ou seja, geração das cavas pertencentes à fronteira do Pareto-Ótimo. Visando simplificar a estrutura da base de dados relativos ao modelo de blocos, para o cálculo das massas dos blocos, admitiu-se que todos os blocos possuem a mesma densidade e, portanto, possuem a mesma massa cujo valor considerado foi o de uma unidade de peso.

Com relação aos teores, também com o objetivo de simplificar a geração dos modelos de blocos a serem trabalhados, os valores de teor médio foram calculados a partir dos valores econômicos na razão de 10/1, ou seja, para os blocos com valores econômicos positivos, os valores de teor médio terão dez vezes menos o valor econômico. Por exemplo: um bloco de 12 unidades monetárias terá um teor médio de 1,2%.

Abaixo algoritmo utilizado para geração do arquivo ASCII referente ao modelo de blocos.

```
Para i = 1 até NLinhas
  Para j = 1 até NColunas
    Para k = 1 até NNiveis
      ValorBloco = CalculaValorBloco(k, j)
      se ValorBloco >= 0 então
        TeorBloco = ValorBloco / 10
      senão
        TeorBloco = -1
      fim-se
      Escreve arquivo (i, j, K, ValorBloco, TeorBloco)
    fim-para
  fim-para
fim-para
```

3.4 Codificação das rotinas computacionais para os problemas propostos

Os algoritmos foram desenvolvidos em *Microsoft Visual Basic* (*VBA 6.3*) disponível no aplicativo *Microsoft Excel*, possibilitando uma maior divulgação dos conceitos envolvidos neste trabalho. É claro que tal opção desfavorece uma aplicação prática real, uma vez que geralmente tem-se modelos de blocos de grandes dimensões que, por sua vez, demandam altos custos computacionais. Também foram codificados algoritmos em *Delphi* (*V3.0 - Borland International*) uma vez que esse compilador tem desempenho superior ao do *VBA*.

Foram implementados três algoritmos genéticos. Destinou-se o primeiro à resolução de problemas mono-objetivos. O segundo, à solução dos problemas multiobjetivos. Ambos codificado em *VBA*. Já o terceiro, codificado em *Delphi*, trabalha tanto para o problema mono-objetivo quanto para os multiobjetivos. Os anexos do item 8 contêm as listagens dos algoritmos codificados.

Para o primeiro programa de computador desenvolvido visando a resolução dos problemas mono-objetivos, utilizou-se a estrutura do algoritmo genético simples onde a população inicial foi gerada aleatoriamente. O processo de seleção utilizado foi o do método da roleta, cruzamento de um ponto e mutação. Para o processo de elitismo foi implementada uma variante que possibilita a definição de uma probabilidade escolhida pelo tomador de decisão que copia os melhores indivíduos no processo de seleção.

A codificação do algoritmo multiobjetivo tve por base o algoritmo desenvolvido por Dias (2000) em *Matlab*, disponível na página do professor Dr. João Antônio de Vasconcelos, que faz parte do Curso de Pós-Graduação em Engenharia Elétrica da UFMG cujo endereço eletrônico é www.cpdee.ufmg.br/~joao no link da disciplina otimização multiobjetivo.

Para o problema em pauta, cada cromossomo foi representado pelo conjunto de blocos de valores positivos contidos no modelo de blocos que serão lavrados considerando-se as restrições geotécnicas. Nesse caso, o cromossomo será do tamanho do número de blocos de minério que existe no modelo de blocos e corresponde a um indivíduo que é um candidato à solução do problema.

3.5 Validação dos algoritmos

Uma vez que os trabalhos iniciais foram desenvolvidos no modelo apresentado por Lerchs e Grossmann (1965), para o caso de maximização do lucro não descontado e reservas, já se conhece a solução. Os resultados são apresentados no item 5.

Para os trabalhos de otimização multiobjetivo as rotinas computacionais desenvolvidas foram validadas por funções-teste disponíveis na literatura. Os resultados são apresentados no item 6.

4 Descrição dos algoritmos e programas de computador codificados

4.1 Códigos implementados

Conforme descrito no item 2.1, trata-se de um problema de maximização. A partir do modelo de blocos contendo um valor monetário para cada bloco, o objetivo é maximizar o valor total dos blocos extraídos, tendo em vista que existem blocos com valores positivos e negativos e, ainda, as restrições geomecânicas. Como resultado, tem-se o valor máximo que se pode obter da cava, bem como sua conformação final, ou seja, a geometria final da cava, conforme ilustrado na figura 2.9.

Ressalta-se que, nesse modelo, existem 36 blocos com valores positivos, que foram trabalhados pelo algoritmo, uma vez que o mesmo faz a busca somente destes blocos. A extração dos blocos negativos é conseqüência do ângulo geral de talude aplicado que, no caso, foi de 45 graus.

Depois de devidamente validadas as rotinas computacionais desenvolvidas, foram realizadas várias simulações visando obter os resultados gerados para os seis tipos de problemas propostos.

Foram executados testes no algoritmo genético simples com elitismo para o problema de maximização do lucro, segundo os dados descritos no item 2.1 - algoritmo de Lerchs-Grossmann em 2D.

Já para o programa baseado no modelo NSGA, foram realizados testes em problemas multiobjetivos disponíveis na literatura: Schafferf2 e Schafferf3. O mesmo foi utilizado para otimização do lucro não descontado e para os problemas multiobjetivo de maximização de reservas e lucro não descontado, maximização de recursos e teor médio e, finalmente, maximização do lucro não descontado e teor médio.

4.2 Descrição do algoritmo genético simples (AGS) com elitismo em *VBA*

4.2.1 *Características Gerais*

O programa de computador *Evolpit* desenvolvido em *VBA* na plataforma *Windows*, trabalha a partir de um arquivo *ASCII* que contem informações relativas a um recurso mineral que, por sua vez, é representado por um modelo de blocos.

O modelo de recursos utilizado é uma adaptação do modelo apresentado por L&G onde se pode trabalhar com até 22 seções totalizando até 6.776 blocos (22 linhas, 22 colunas e até 14 níveis).

A função de mérito (objetivo) inserida no algoritmo foi a de maximização do valor não descontado da cava final (otimização mono objetivo).

Visando a aplicação de parte da teoria que engloba os algoritmos genéticos, existem os seguintes recursos no código do algoritmo implementado:

Parâmetros de entrada:

- Modelo de blocos:
    - Origem;
    - Tamanho dos blocos;
    - Tamanho do modelo de blocos.
- Parâmetros de controle de *loop*:
    - Tamanho da população;
    - Probabilidade de cruzamento;
    - Probabilidade de mutação;
    - Percentual de elitismo;
    - Número da semente.

Resultados:

- Visualização das informações do modelo de blocos processado;

- Resultados de todos os indivíduos da população por geração;

- Visualização dos blocos lavrados através de seções verticais;

- Resumo dos resultados no caso de processamento de mais de um cenário de lavra.

4.2.2 *Interface de parâmetros*

Assim que é carregada na memória, a interface responsável pela entrada dos parâmetros a serem utilizados pelo programa é exibida conforme figura a seguir:

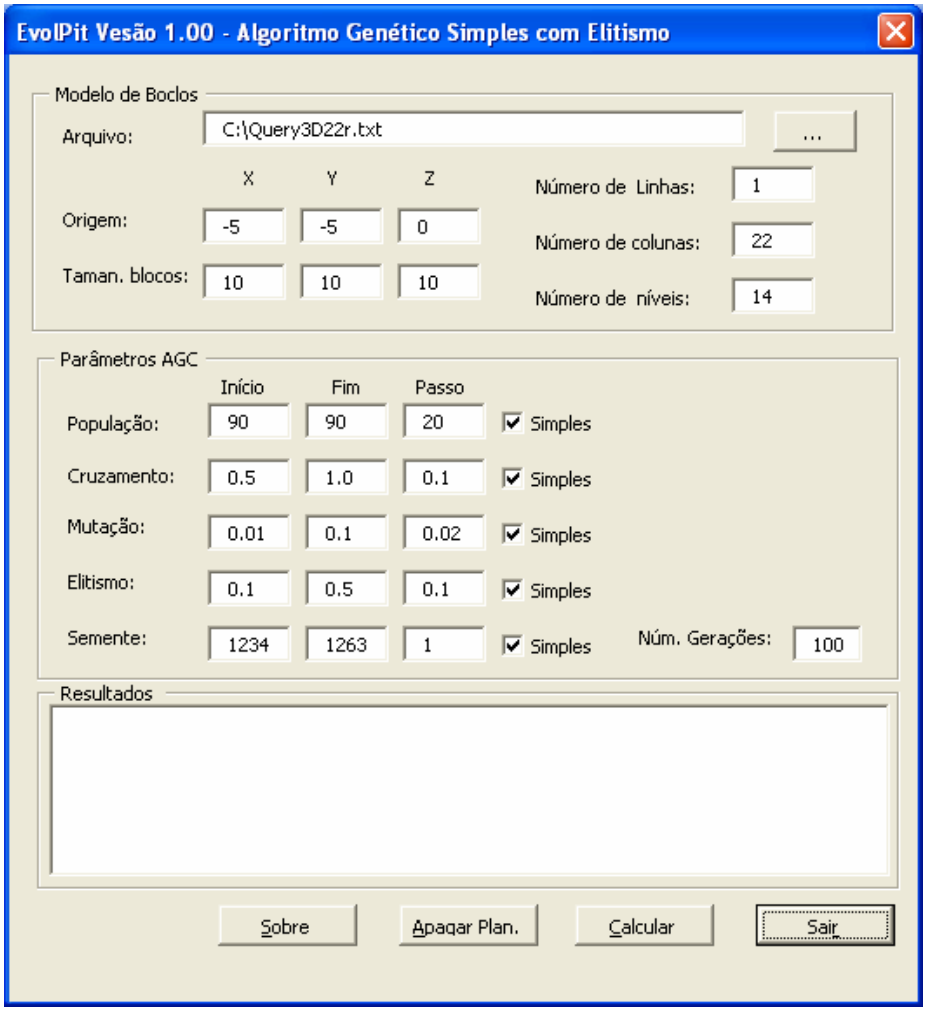


Figura 4.2.2.1 – Interface AGS (*VBA*).

4.2.2.1 *Modelo de Blocos*

Origem: definem-se as coordenadas x,y e z que correspondem ao vértice do canto superior anterior esquerdo do paralelepípedo que representa o modelo de recursos em blocos discretizados de mesmo tamanho.

Tamanho dos Blocos: refere-se aos tamanhos de cada bloco regular, nas direções dos três eixos do paralelepípedo (x,y e z). Ressalta-se que todos os blocos possuem o mesmo tamanho, ou seja, o algoritmo não trabalha com sub-blocos.

Tamanho do modelo: define as dimensões do paralelepípedo relativo ao modelo de blocos nas três direções (linhas, colunas e níveis).

Estrutura do arquivo de modelo de blocos: Arquivo ASCII separado por vírgulas contendo seis colunas, necessariamente na seguinte ordem:

1) Número da linha de cada bloco do modelo;

2) Número da coluna de cada bloco do modelo;

3) Número do nível de cada bloco do modelo;

4) Valor monetário do bloco (poderá ser positivo ou negativo).

Salienta-se que, na primeira linha, devem existir as descrições de cada coluna do arquivo.

4.2.2.2 *Parâmetros de controle de loop*

Semente: parâmetro utilizado para criação de números aleatórios no algoritmo. Esse parâmetro é importante para se obter repetibilidade dos cenários avaliados.

População: tamanho da população que será utilizada no algoritmo genético. O tamanho máximo da população permitido nesse código é de 2000 indivíduos.

Cruzamento: refere-se à taxa de cruzamento que será utilizada no algoritmo genético. Geralmente varia de 0,5 a 1,0.

Mutação: refere-se à taxa de mutação que será utilizada no algoritmo genético. Geralmente varia de 0,001 a 0.1.

Elitismo: será a taxa utilizada para cópia do melhor indivíduo de cada geração. Por exemplo, uma taxa de 0,2 em uma população de 100 indivíduos: serão copiados 20 indivíduos iguais ao melhor indivíduo da geração. Mesmo com uma taxa de zero, o algoritmo sempre manterá o melhor indivíduo em cada geração.

4.2.3 *Resultados*

Após o processamento dos cálculos, são geradas várias planilhas conforme descrito a seguir:

4.2.3.1 *Modelo Lido*

A planilha *Model_CSV* contem os parâmetros referentes à posição geométrica dos blocos (linha, coluna e nível), bem como ao valor monetário de cada bloco.

4.2.3.2 *Visualização seções verticais*

Serão geradas tantas planilhas de linhas quantas existirem no arquivo de modelo de blocos lido. Terão o nome *r_nome* da linha, por exemplo: *r1* que representa a linha (*row*) 1. A figura a seguir exibe a seção vertical referente à linha 1 de um modelo de blocos de apenas uma linha:

-4	-4	-4	-4	-4	8	12	12	0	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	0	12	12	8	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	8	12	12	0	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	0	12	12	8	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	8	12	12	0	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	0	12	12	8	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	8	12	12	0	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	0	12	12	8	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	8	12	12	0	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4

Figura 4.2.3.2.1 – Modelo de blocos lido AGS (*VBA*).

4.2.3.3 *Planilhas auxiliares*

PopIni: contem os dados referentes à população inicial cujos valores podem ser verdadeiro (extrair bloco) ou falso (não extrair bloco).

PopFit: contem os valores da aptidão calculada para cada individuo para todas as gerações.

NCopias: contem o número de cópias de cada indivíduo calculado pelo algoritmo genético de acordo com a sua aptidão.

Resultados Grupo: Caso sejam utilizadas algumas opções de loop, o resultado dos melhores indivíduos de cada condição simulada será armazenado nessa planilha.

4.2.3.4 *Planilha de Resultados*

Nessa planilha, cada célula representa um indivíduo, ou seja, uma cava final. O valor contido em cada célula representa o valor não descontado do lucro da cava. Cada linha representa uma geração. Assim, o número de linhas da planilha será igual ao número de gerações definido na interface de parâmetros. O número de colunas será equivalente ao tamanho da população utilizado.

4.2.3.5 *Seções verticais cava final*

A geometria final da cava final será visualizada por meio de seções verticais conforme a figura a seguir:

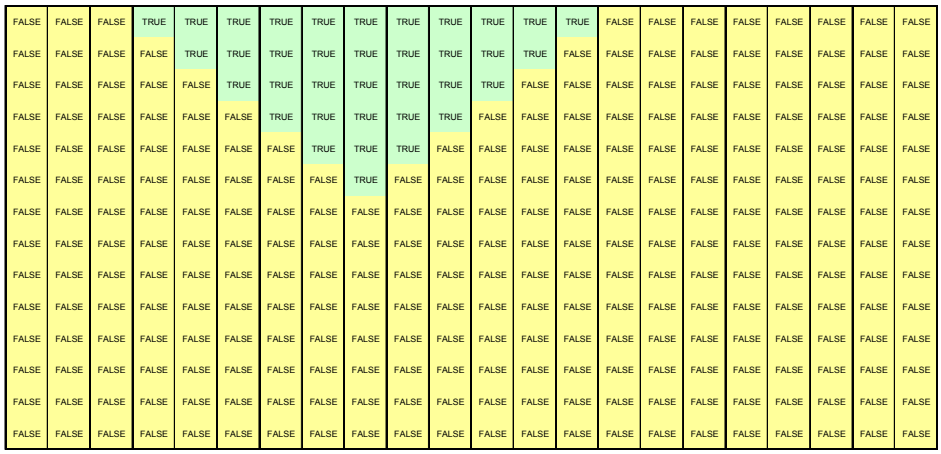


Figura 4.2.3.5.1 – Seção vertical AGS (*VBA*).

Nessa figura, cada célula representa um bloco. Os blocos que serão lavrados estão na cor verde e possuem valor verdadeiro (*true*). Já os blocos em amarelo representam os blocos que permanecerão *in situ* e possuem valor falso (*false*).

4.3 Descrição programa NSGA - *VBA*

4.3.1 Características Gerais

Conforme citado anteriormente, a codificação do algoritmo multiobjetivo teve por base o algoritmo desenvolvido por Dias (2000) em *Matlab*, disponível na página do professor Dr. João Antônio de Vasconcelos, que faz parte do Curso de Pós-Graduação em Engenharia Elétrica da UFMG cujo endereço eletrônico é www.cpdee.ufmg.br/~joao no link da disciplina otimização multiobjetivo.

O objetivo da elaboração dessa planilha que contem o código *VBA* foi validar a migração do código em *Matlab* para o *Visual Basic*, por meio de duas funções de teste disponíveis na literatura.

4.3.2 Descrição das funções de teste utilizadas

As funções testadas são denominadas Schafferf2 e Schafferf3 cujos gráficos no espaço das variáveis são apresentados a seguir:

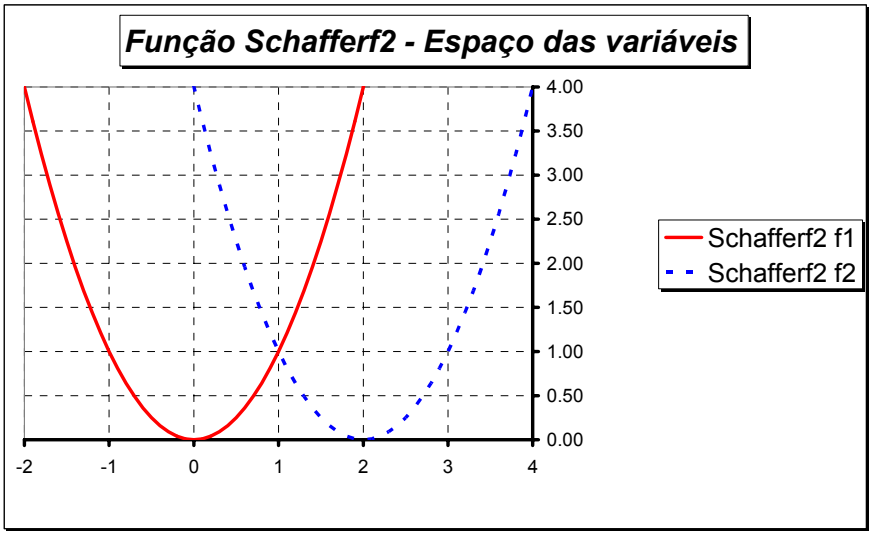


Figura 4.3.2.1 – Gráfico da função Schafferf2 no espaço das variáveis.

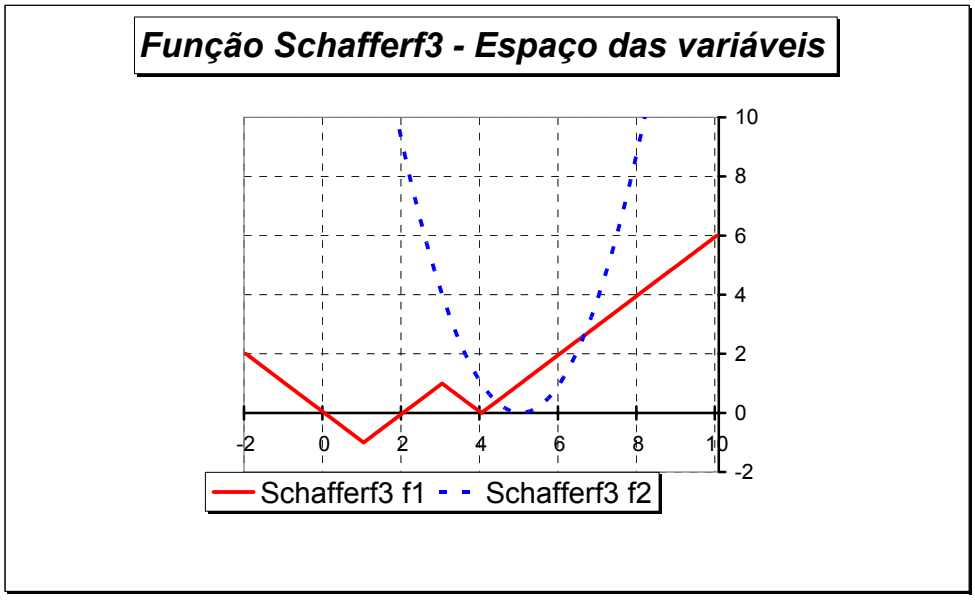


Figura 4.3.2.2 – Gráfico da função Schafferf3 no espaço das variáveis.

Os gráficos a seguir exibem as funções Schafferf2 e Schafferf3 no espaço dos objetivos:

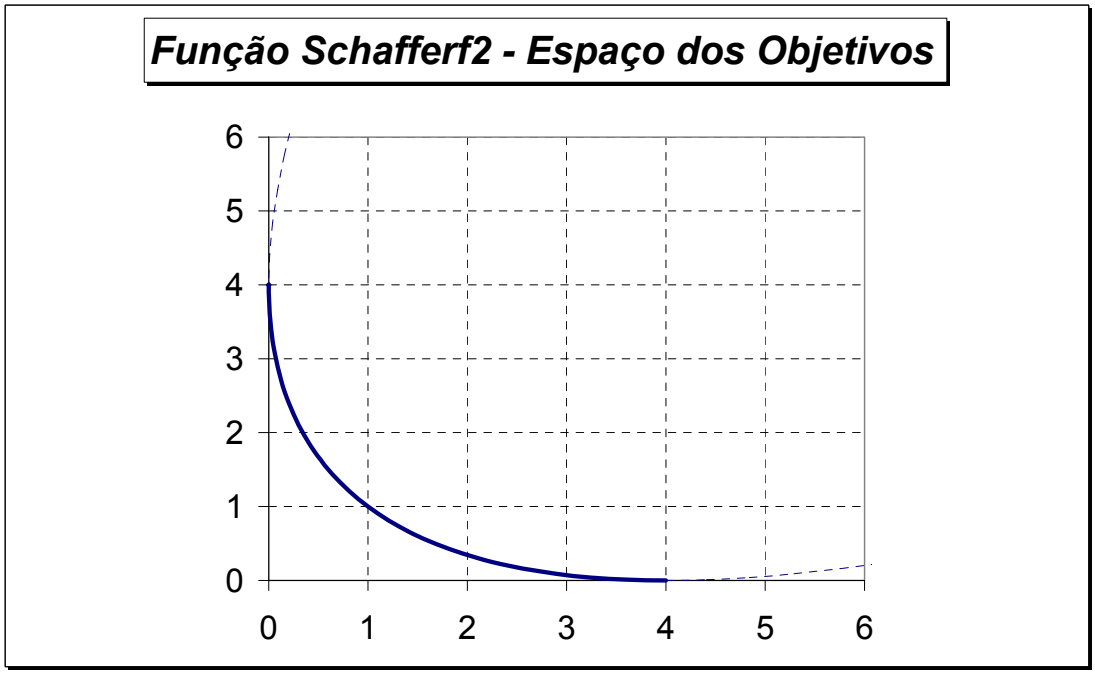


Figura 4.3.2.3 – Gráfico da função Schafferf2 no espaço dos objetivos.

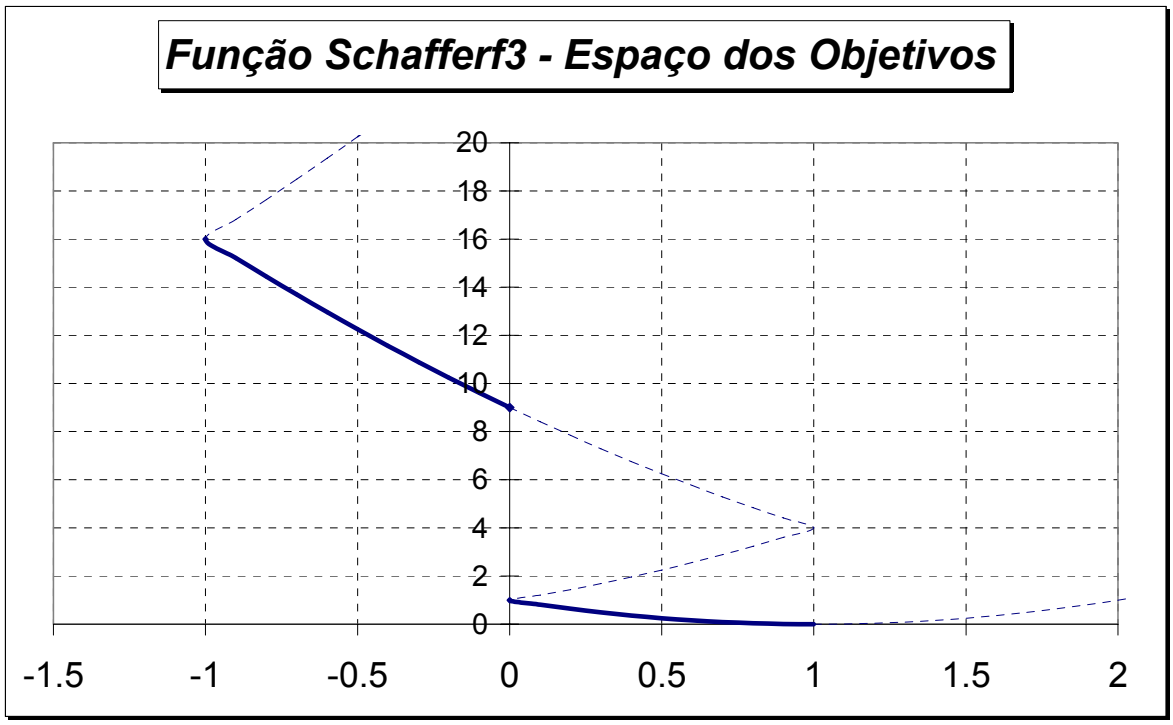


Figura 4.3.2.4 – Função Schafferf3 no espaço dos objetivos.

4.3.3 Parâmetros de entrada

A figura abaixo exibe a interface do programa NSGA II codificado em *Excel – VBA*:

Parâmetros de entrada:	
Tamanho da população:	100
Tamanho da população externa (elitismo):	200
Número máximo de gerações:	120
Problemas: (1) Schafferf2; (2) = Schafferf3	2
Probabilidade de cruzamento [0,1]:	0.9
Probabilidade de mutação [0,1]:	0.01
Semente:	12345
Percentagem de elitismo [0,1]:	0.9
Fator de partilha: (0) automático; (valor personalizado):	0
Limite mínimo Schafferf2	-12
Limite máximo Schafferf2	12
Limite mínimo Schafferf3	-2
Limite máximo Schafferf3	10

Figura 4.3.3.1 – Interface NSGA (*VBA*).

4.3.4 Resultados

Os resultados para as duas funções de teste são disponibilizados por meio dos recursos gráficos disponíveis no *MS Excel* conforme a figura abaixo:

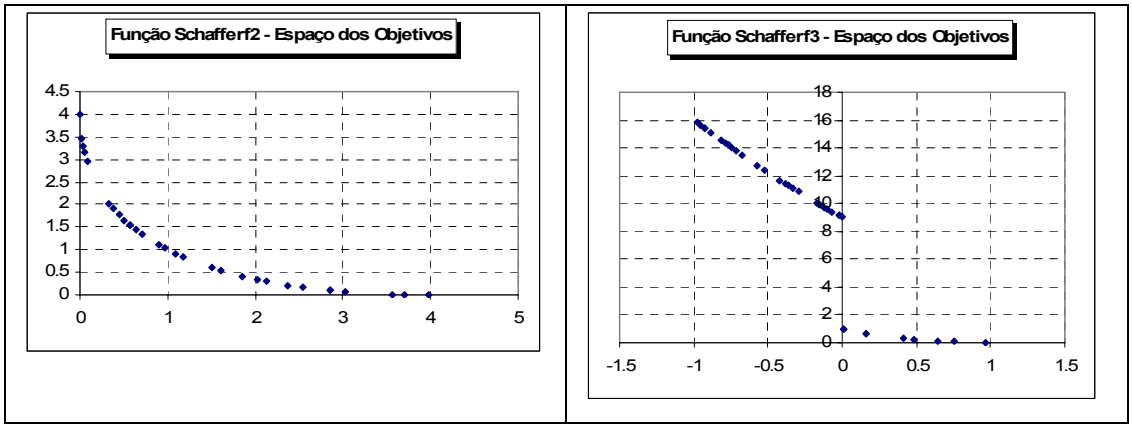


Figura 4.3.4.1 – Resultados gráficos NSGA (*VBA*).

4.4 Descrição do programa *EvolPit - Delphi*

4.4.1 *Características gerais*

O programa de computador *Evolpit* desenvolvido em *Delphi* na plataforma *Windows*, trabalha a partir de um arquivo *ASCII* que contem informações relativas a um recurso mineral que, por sua vez, é representado por um modelo de blocos. O objetivo do programa é avaliar a utilização dos algoritmos genéticos na geração de cavas finais e seqüenciamentos matemáticos em minas a céu aberto.

O modelo de recursos utilizado é uma adaptação do modelo apresentado por L&G onde se pode trabalhar com até 999 seções totalizando até 307.692 blocos (999 linhas, 22 colunas e até 14 níveis).

As funções de mérito (objetivo) inseridas no algoritmo foram a de maximização do valor não descontado da cava final (otimização mono objetivo), e três opções de otimização multiobjetivo: a) Valor & massa minério, b) Valor & teor e c) Massa minério & teor.

Utilizando-se das opções de restrições de massa (número de blocos) e teores (mínimo e máximo) disponíveis na interface do programa, pode-se trabalhar com a construção de cenários de seqüenciamentos de lavra.

Visando a aplicação de parte da teoria que engloba os algoritmos genéticos, existem os seguintes recursos no código do algoritmo implementado:

Parâmetros de entrada:

- Modelo de blocos;

- Opções de otimização;

- Geração da população inicial;

- Parâmetros de controle de *loop;*

- Parâmetros de nicho;

- Opções de cruzamento e adaptação dinâmica;

- Opções de mutação e seleção;

- Opções de restrições.

Resultados:

- Visualização das informações do modelo de blocos processado;

- Histogramas do número de blocos com valores não negativos;

- Resultados de todos os indivíduos da população por geração (Valor e Aptidão);

- Visualização dos blocos lavrados por meio de seções verticais;

- Gráficos de evolução do AG;

- Resumo dos resultados no caso de processamento de mais de um cenário de lavra;

- Gráfico dos resultados da otimização multiobjetivo (Objetivo1 x Objetivo2).

4.4.2 *Parâmetros de entrada*

4.4.2.1 *Interface dos parâmetros de entrada*

Assim que é carregada na memória, a interface responsável pela entrada dos parâmetros a serem utilizados pelo programa está na janela de Parâmetros conforme afigura a abaixo:

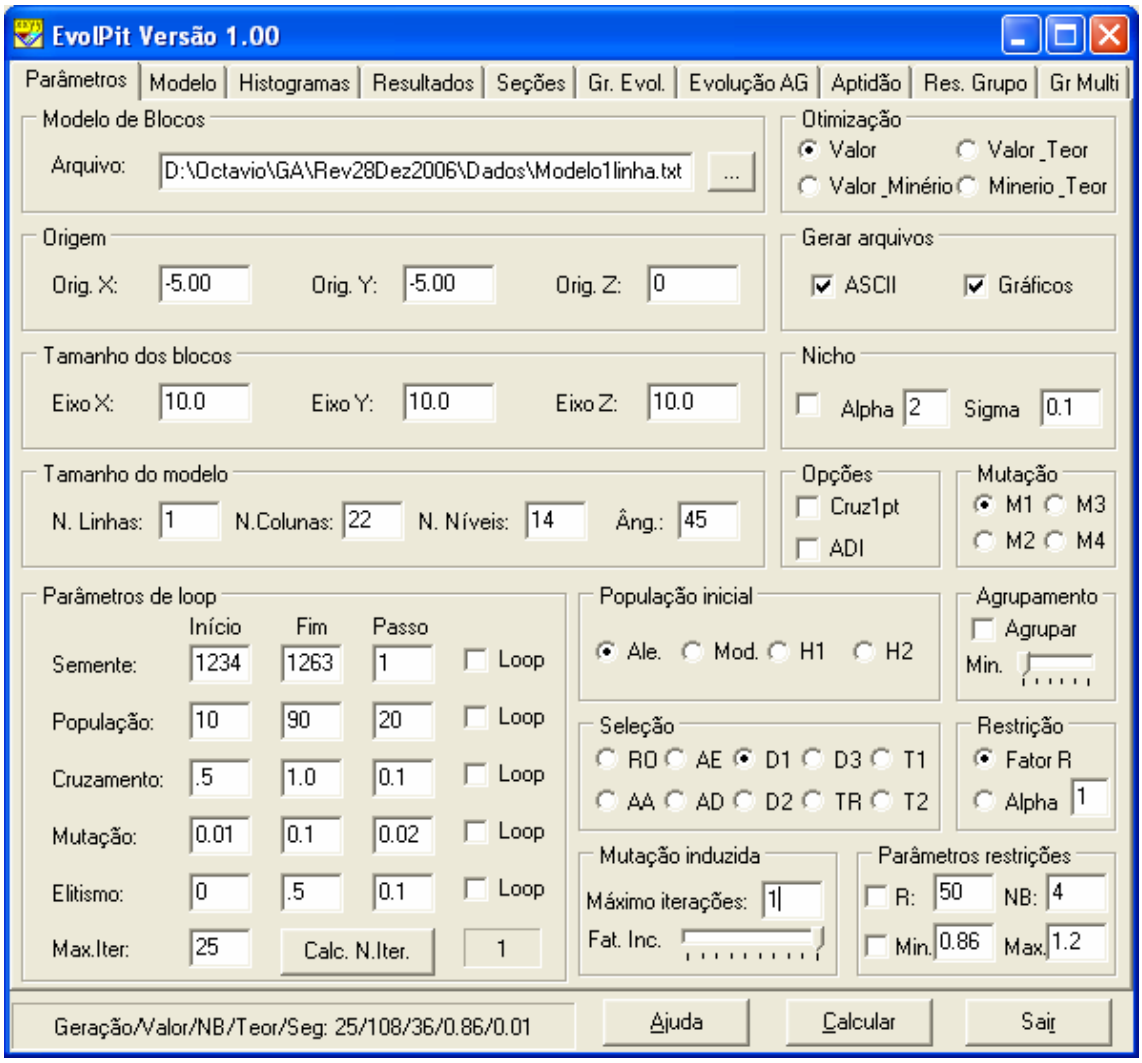


Figura 4.4.2.1 – Interface parâmetros de entrada (*Evolpit*).

4.4.2.2 *Modelo de Blocos*

As definições de origem, tamanho dos blocos e tamanho do modelo são as mesmas do AGS com elitismo codificado em *VBA* descritas no item 4.2.2.1.

Estrutura do arquivo de modelo de blocos: Arquivo ASCII separado por vírgulas contendo seis colunas, necessariamente na seguinte ordem:

1) Número da linha de cada bloco do modelo;

2) Número da coluna de cada bloco do modelo;

3) Número do nível de cada bloco do modelo;

4) Valor monetário do bloco (poderá ser positivo ou negativo);

5) Teor médio do bloco;

6) Valor “0” ou “1” que representa a população inicial. O zero representa blocos não lavrados e o valor igual à unidade blocos dentro da cava final.

Salienta-se que, na primeira linha, devem estar as descrições de cada coluna do arquivo.

4.4.2.3 *Opções de Otimização*

Utilizando-se aa interface do programa, pode-se optar por quatro tipos de função de otimização:

- Valor: que corresponde à maximização do valor não descontado da cava final;

- Valor_Minério: otimização multiobjetivo onde se quer maximizar o valor não descontado da cava final e os recursos, ou seja, o número de blocos não negativos;

- Valor_Teor: otimização multiobjetivo onde se quer maximizar o valor não descontado da cava final e o teor médio;

- Minério_Teor: otimização multiobjetivo onde se quer maximizar o número de blocos não negativos e o teor médio.

4.4.2.4 *Geração da população inicial*

Aleatória: o processo de seleção dos blocos de minério que serão lavrados é aleatório, ou seja, independe do valor e da posição espacial do bloco dentro do modelo.

Modelo: os blocos de minério que serão lavrados na população inicial são os que possuem o valor “1” no arquivo de modelo de blocos na sexta coluna do arquivo.

Opções de Agrupamento: Consistem em agrupar os blocos positivos que estejam aflorando na superfície, ou seja, sem blocos negativos bloqueando os mesmos. Além disso, os cones contendo somente blocos de minério também são agrupados. A opção de agrupar trabalha em conjunto com a barra de agrupamento, logo abaixo da mesma, na interface do programa. Quando essa opção está marcada além da posição de mínimo (Min.), a seguinte expressão é utilizada para agrupar os cones dos blocos:

round((InicioEsteril + k)*AG.FCluster/10)-1 (4.4.2.4.1)

onde,

InicioEsteril: corresponde à posição da lista de cones fortes (aflorantes + somente blocos positivos) mais um;

k: posição na lista em que, após ordenação da mesma, o valor da cava foi máximo;

AG.FCluster: Fator de agrupamento que varia de 0 a 5 de acordo com a posição selecionada na interface.

Heurística 1 (H1): o processo de seleção dos blocos de minério que serão lavrados na população inicial é baseado no algoritmo seguinte:

```
VMax = Populacao.Valor
troca = verdadeiro
para k = 1 até AG.FCluster + 1 faça
  para i = InicioEsteril até Len_Cromos faça
    Populacao.Cromos[i] = Troca
    Avalia_Individuo
    se Populacao.Valor >= VMax então
      VMax = População.Valor
    senão
      Populacao.Cromos[i] = não Troca
    fim-se
  fim-para;
  Troca = não Troca
fim-para;
```

A idéia básica desse algoritmo é lavrar os blocos que agregam valor à cava final simplesmente avaliando se a extração do bloco aumenta o valor total da cava.

Heurística 2 (H2): o processo de seleção dos blocos de minério que serão lavrados na população inicial é baseado no seguinte algoritmo:

```
para i = IEsteril to Len_Cromos faça
  Populacao[2].Cromos[i] = falso
  Populacao[1].Cromos[i] = verdadeiro
  Avalia_Individuo(1)
  ConeMin[i] = Populacao[1].Valor
  Para j = IEsteril até Len_Cromos faça
    Populacao[1].Cromos[j] = falso
  Fim-para
Fim-para
QuickSortDM(IEsteril,Len_Cromos)
k = 0
para i = IEsteril até IEsteril + AG.FCluster - 1 faça
  k = k +1
  para j = 1 até AG.SizePop faça
    Populacao[j].Cromos[i] = verdadeiro
  Fim-para
Fim-para
IEsteril = IEsteril + k
```

Nessa heurística faz-se a ordenação de todos os blocos positivos de acordo com o seu valor monetário e, em seguida, lavram-se os blocos de acordo com o fator de agrupamento definido na interface.

4.4.2.5 *Parâmetros de controle de loop*

Os parâmetros s*emente, população cruzamento, mutação e elitismo* são os mesmos utilizados no algoritmo genético simples com elitismo conforme descrito no item 4.2.2.2.

Botão Calc N.Iter.: utilizado para calcular o número de cenários que serão processados pelo algoritmo, de acordo com os parâmetros de *loop* definidos na interface.

4.4.2.6 *Parâmetros de nicho*

Alpha: corresponde ao termo α da equação 2.5.1.1 e tem por objetivo penalizar os indivíduos mais próximos.

Sigma: corresponde ao termo *βshared* da equação 2.5.1.1 e indica o grau de vizinhança entre dois indivíduos.

4.4.2.7 *Opções de cruzamento e adaptação dinâmica*

Cruz1pt: nessa opção, o processo de cruzamento do algoritmo genético será realizado com apenas um ponto. Caso essa opção seja desmarcada, o algoritmo utilizará o cruzamento com dois pontos.

ADI (Adaptação Dinâmica por indivíduo): Conforme descrito no item 2.4.6, as probabilidades de cruzamento e mutação, para cada indivíduo, variam de acordo com o seu desempenho e com os valores de desempenho médio e máximo encontrados na população. Nesse critério, deseja-se evitar que as boas soluções se percam com facilidade, e que as más sejam totalmente destruídas.

4.4.2.8 *Opções de mutação e seleção*

Foram implementados 10 diferentes tipos de seleção, 4 tipos de mutação e ainda uma opção denominada mutação induzida.

A seguir, a lista dos 10 diferentes métodos de seleção implementados no programa:

RO – Método da Roleta:

```
Para i = 1 até AG.SizePop faça
  NAleat = Random
  Para j = 1 até AG.SizePop - 1 faça
    se (NAleat >= ProbPopAc[j]) E
       (NAleat<ProbPopAc[j+1])
      então
        NCopias[j] = NCopias[j] + 1
        NC = NC + 1
    Fim-se
  Fim-para
Fim-para
se NC < AG.SizePop então
  para i = NC + 1 até AG.SizePop faça
    NCopias[i] = NCopias[i] + 1
  Fim-para
Fim-se
```

AA – Amostragem Aleatória:

```
Para i = 1 até AG.SizePop faça
  j = round((AG.SizePop -1) * Random) + 1
  NCopias[j] = NCopias[j] + 1
  NC = NC + 1
fim-para
se NC < AG.SizePop então
  para i = NC + 1 até AG.SizePop faça
    j = round((AG.SizePop -1) * Random) + 1
    NCopias[j] := NCopias[j] + 1
  fim-para
fim-se
```

AE – Amostragem Estocástica:

```
NC = 0
para i = 1 até AG.SizePop faça
   NCopias[i] :=
       trunc(Populacao[i].Fitness/(AcumAP/AG.SizePop))
   se NCopias[i] > 0 então NC := NC + 1
fim-para
enquanto NC <= AG.SizePop faça
  para i = 1 até AG.SizePop faça
    NAleatorio =  Random
    se (Populacao[i].Fitness/Media-
       trunc(Populacao[i].Fitness/Media))/(AG.SizePop-
```

```
            NC+1)>NAleatorio
 então
        NCopias[i] := NCopias[i] + 1;
        NC := NC + 1;
    Fim-se
    if NC >= AG.SizePop então Sair
  fim-para
fim-enquanto
```

AD – Amostragem Determinística:

```
NC = 0
para i = 1 até AG.SizePop faça
   NCopias[i] = trunc(Populacao[i].Fitness/Media)
   NC = NC + NCopias[i]
fim-para
enquanto NC <= AG.SizePop faça
  j = Round((AG.SizePop - 1) * Random) + 1
  se ((Populacao[j].Fitness / Media) -
      trunc(Populacao[j].Fitness / Media)) > Random
  então
    NC = NC + 1
    NCopias[j] = NCopias[j] + 1
  fim-se
fim-enquanto
```

D1 – Amostragem Determinística Variante 1:

```
para i = 1 até AG.SizePop faça
  NCopias[i] = trunc(Populacao[i].Fitness / Media)
fim-para
```

D2 – Amostragem Determinística Variante 2:

```
NC = 0
para i = 1 até AG.SizePop faça
  NCopias[i] = trunc(Populacao[i].Fitness / Media)
  NC = NC + NCopias[i]
fim-para
k = 1
enquanto NC <= AG.SizePop faça
  k = k - 0.1
  para i := 1 até AG.SizePop faça
  se ( (Populacao[i].Fitness / Media) -
       trunc(Populacao[i].Fitness / Media) ) > k
  então
    NC = NC + 1
    NCopias[i] = NCopias[i] + 1
    se NC >= AG.SizePop então sair
  fim-se
```

```
fim-enquanto
```

D3 – Amostragem Determinística Variante 3:

```
NC = 0
para i = 1 até AG.SizePop faça
  NCopias[i] = trunc(Populacao[i].Fitness / Media)
  NC = NC + NCopias[i]
Fim-para
enquanto NC <= AG.SizePop faça
  para i = 1 até AG.SizePop faça
    se NCopias[i] > 0 então
      NC = NC + 1
      NCopias[i] = NCopias[i] + 1
      if NC >= AG.SizePop então sair
    fim-se
  fim-para
fim-enquanto
```

TR – Método de Torneio:

```
NC = 0
para i = 1 até AG.SizePop -1 faça
  se Populacao[i].Fitness > Populacao[i+1].Fitness
    então
      NCopias[i] = NCopias[i] + 1
    senão
      NCopias[i+1] = NCopias[i+1] + 1
  Fim-se
  NC = NC + 1
Fim-para
se NC < AG.SizePop então
  para i = NC + 1 até AG.SizePop faça
    j = round((AG.SizePop -1) * Random) + 1
    NCopias[j] = NCopias[j] + 1
  Fim-para
Fim-se
```

T1 – Método de Torneio Variante 1:

```
para i = 1 até AG.SizePop - 1 faça
  se Populacao[i].Fitness > Populacao[i + 1].Fitness
então
      NCopias[i] := NCopias[i] + 1
    senão
      NCopias[i + 1] := NCopias[i + 1] + 1
  Fim-se
Fim-para
k = 1
enquanto NC <= AG.SizePop faça
  k = k - 0.1
  para i = 1 até AG.SizePop faça
  se ( (Populacao[i].Fitness / Media) -
       trunc(Populacao[i].Fitness / Media) ) > k
    então
      NC := NC + 1;
      NCopias[i] := NCopias[i] + 1;
      Se NC >= AG.SizePop então sair
  fim-se
fim-enquanto
```

T2 – Método de Torneio Variante 2:

```
NC = 0
enquanto NC <= AG.SizePop faça
  NC = NC + 1
  j = Round((AG.SizePop - 1) * Random) + 1
  i = Round((AG.SizePop - 1) * Random) + 1
  se Populacao[j].Fitness > Populacao[i].Fitness então
    se Random < 0.75 então
      NCopias[j] := NCopias[j] + 1
    senão
      NCopias[i] := NCopias[i] + 1
    fim-se
  fim-se
fim-enquanto
```

A seguir, a lista dos 4 tipos de mutação ora implementados:

M1 – Heurística de Mutação 1

```
PM = AG.PMut
para i = 2 até AG.SizePop faça
  para j = InicioEsteril até Len_Cromos faça
    se Random <= PM então
      se Populacao[i].Cromos[j] = True então
        Populacao[i].Cromos[j] = False
      senão
        Populacao[i].Cromos[j] = True
      fim-se
    fim-se
  fim-para
fim-para
```

M2 – Heurística de Mutação 2

```
se AG.PMut * Len_Cromos < 1 então
  para i = 2 até AG.SizePop faça
    para j = 1 até Len_Cromos faça
      se Populacao[i].Cromos[j] = True então
          Populacao[i].Cromos[j] = False
      senão
          Populacao[i].Cromos[j] = True
      fim-se
    fim-para
senão
  para z = 1 até round(AG.Pmut*Len_Cromos) faça
    para i = 2 To AG.SizePop faça
      k = round(Random * (Len_Cromos-1))+1
      para j = 1 To k do
        se Populacao[i].Cromos[j] = True então
          Populacao[i].Cromos[j] = False
        senão
          Populacao[i].Cromos[j] = True
      fim-para
    fim-para
  fim-para
fim-se
```

M3 – Heurística de Mutação 3

```
se AG.SizePop * AG.PMut * Len_Cromos < 1 então
  para i = 2 até AG.SizePop faça
    para j = 1 até Len_Cromos faça
      se Random <= AG.PMut Then
        se Populacao[i].Cromos[j] = True então
          Populacao[i].Cromos[j] = False
        senão
          Populacao[i].Cromos[j] = True
        fim-se
  fim-se
senão
  para z=1 até round(AG.SizePop*AG.Pmut*Len_Cromos)
faça
    i = round((AG.SizePop-1) * Random) +1
    se i = 1 então  i = 2
    k = round(Random * (Len_Cromos-1))+1
    para j = 1 até k faça
      se Populacao[i].Cromos[j] = True então
        Populacao[i].Cromos[j] = False
      senão
        Populacao[i].Cromos[j] = True
      fim-se
    fim-para
  fim-para
fim-se
```

M4 – Heurística de Mutação 4

```
para i = 2 até AG.SizePop faça
  PM = AG.PMut*PercMut[i]
  para j = InicioEsteril até Len_Cromos faça
    se Random <= PM então
      se Populacao[i].Cromos[j] = True então
        Populacao[i].Cromos[j] = False
      senão
        Populacao[i].Cromos[j] = True
      fim-se
    fim-se
  fim-para
fim-para
```

Para a opção denominada mutação induzida, foi implementado o seguinte algoritmo:

```
ValorAntigo = Populacao[1].Valor
para j = 1 até AG.NIterMut faça
  para i = 1 até Len_Cromos faça
    Populacao[1].Cromos[i] = not Populacao[1].Cromos[i]
    Avalia_Individuo(1)
    se Populacao[1].Valor <= ValorAntigo então
      Populacao[1].Cromos[i] = not
                               Populacao[1].Cromos[i]
    senão
      ValorAntigo := Populacao[1].Valor
    fim-se
  fim-para
fim-para
```

Essa mutação é aplicada de acordo com a barra de fator de incremento (Opção: Fat.Inc.) da interface que é função do número de gerações. Por exemplo, para 100 Gerações (opção Max.Iter.), com a barra totalmente à direita, serão executadas 10 operações de mutação; daí o nome mutação induzida.

4.4.2.9 *Opções de restrições*

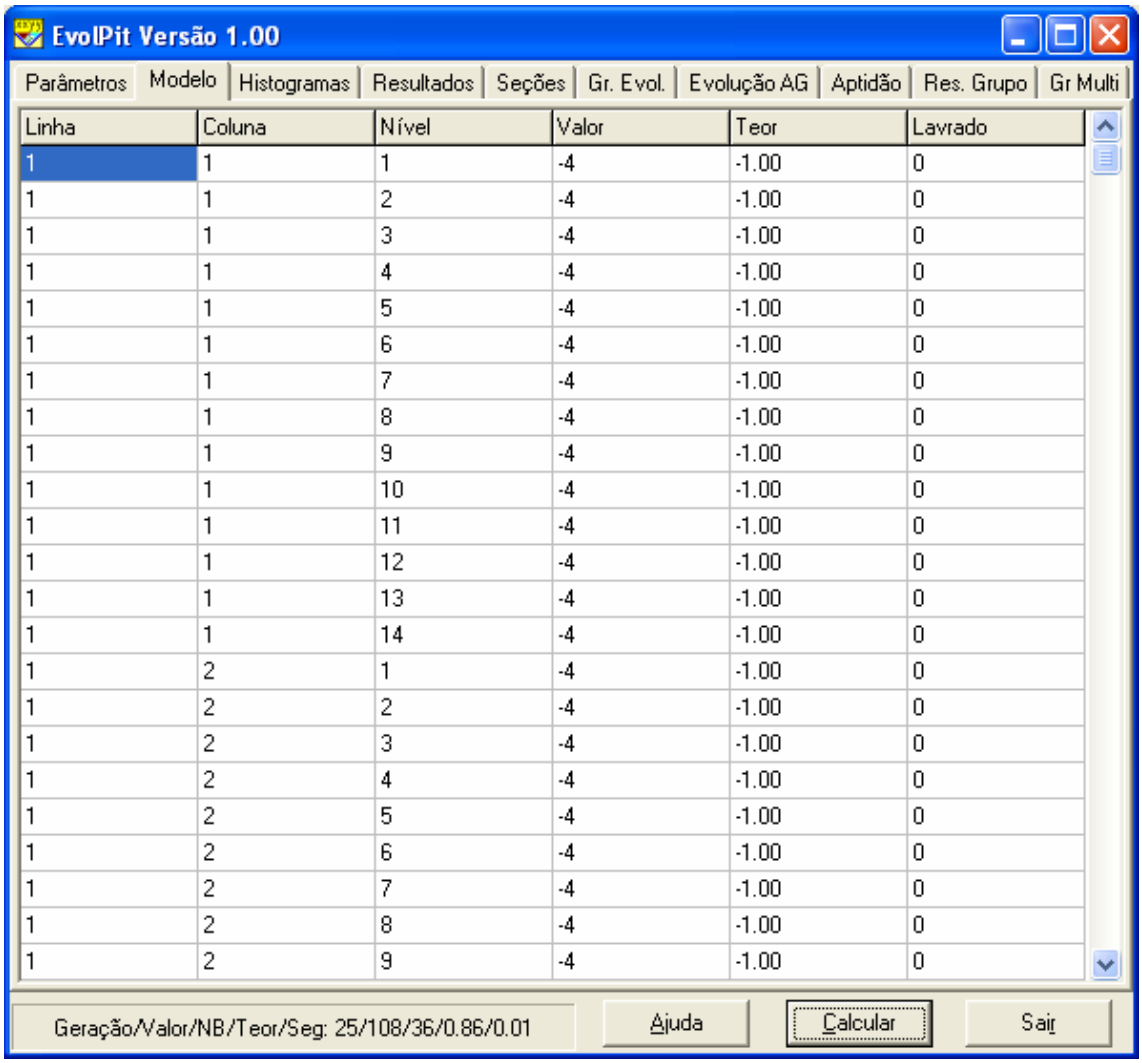
Estão disponíveis, na interface, dois tipos de restrições que podem trabalhar em separado ou em conjunto. A primeira corresponde ao número máximo de blocos que se pode conter na cava final (opção NB). A segunda permite a definição de um intervalo de teores entre um valor mínimo (opção Min.) e um valor máximo (opção Max.).

Uma vez definido(s) o(s) tipo(s) de restrição(ões), deve-se estabelecer como será tratada a restrição na função objetivo, ou seja, o valor do parâmetro R (fator de penalização) e, caso se queira, fazer uso da iteração na função de penalização, conforme descrito em 4.4.2.6 o parâmetro Alpha de restrição.

4.4.3 Resultados

4.4.3.1 *Visualização das informações do modelo de blocos processado*

Uma vez processados os cálculos, as informações relativas ao modelo de blocos processadas pelo programa podem ser visualizadas na janela de Modelo da interface, conforme se segue:



Linha	Coluna	Nível	Valor	Teor	Lavrado
1	1	1	-4	-1.00	0
1	1	2	-4	-1.00	0
1	1	3	-4	-1.00	0
1	1	4	-4	-1.00	0
1	1	5	-4	-1.00	0
1	1	6	-4	-1.00	0
1	1	7	-4	-1.00	0
1	1	8	-4	-1.00	0
1	1	9	-4	-1.00	0
1	1	10	-4	-1.00	0
1	1	11	-4	-1.00	0
1	1	12	-4	-1.00	0
1	1	13	-4	-1.00	0
1	1	14	-4	-1.00	0
1	2	1	-4	-1.00	0
1	2	2	-4	-1.00	0
1	2	3	-4	-1.00	0
1	2	4	-4	-1.00	0
1	2	5	-4	-1.00	0
1	2	6	-4	-1.00	0
1	2	7	-4	-1.00	0
1	2	8	-4	-1.00	0
1	2	9	-4	-1.00	0

Figura – 4.4.3.1.1 Modelo de blocos processado (*Evolpit*).

4.4.3.2 *Histogramas dos blocos de minério*

Na janela histogramas, são construídos três gráficos relativos ao número de blocos com valores positivos no modelo lido por linhas, colunas e níveis, conforme a figura abaixo:

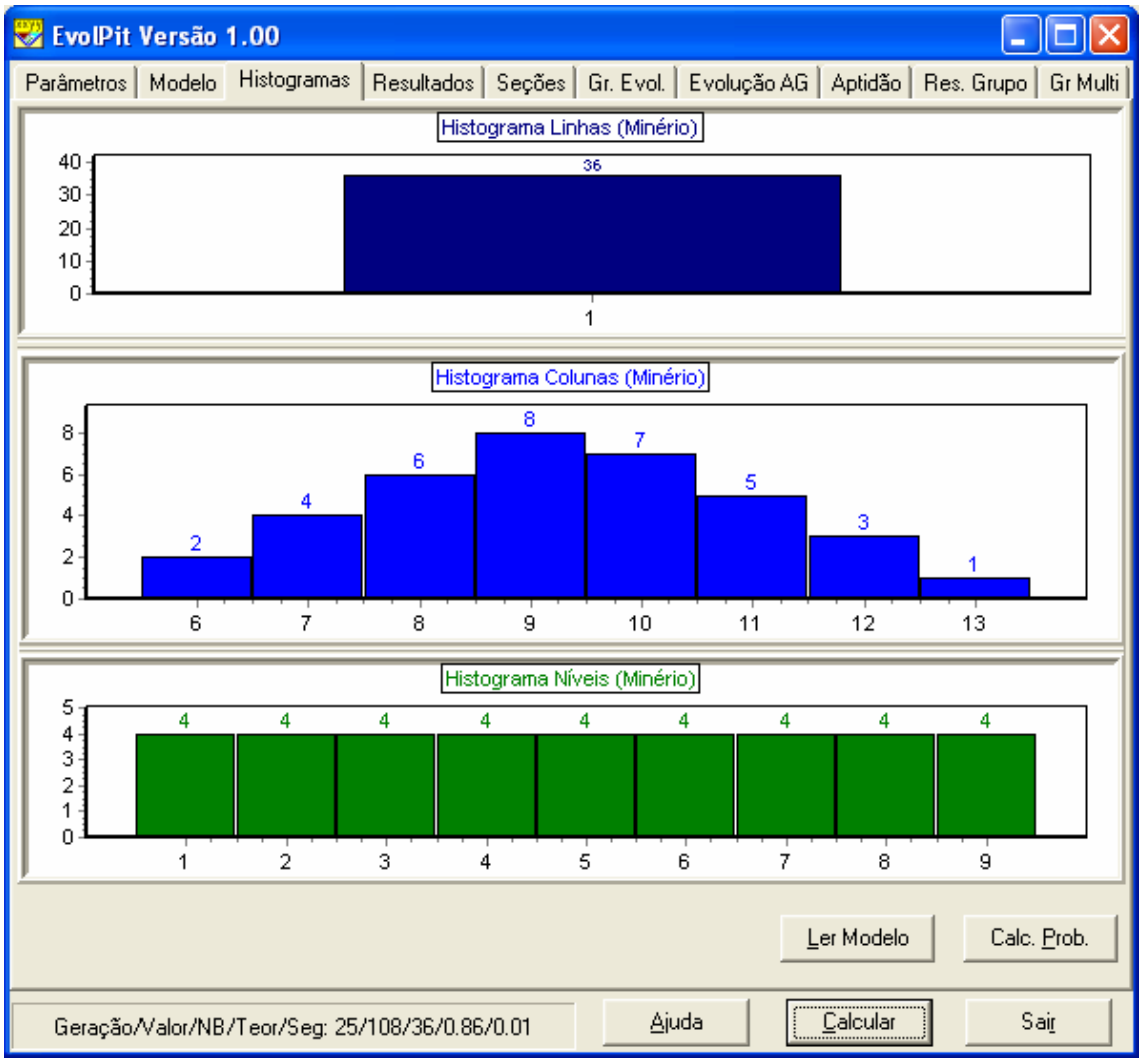


Figura 4.4.3.2 1 – Histogramas dos blocos de valores monetários positivos.

4.4.3.3 *Resultados de todos os indivíduos da população por geração (Valor e Aptidão)*

Na janela Resultados, cada linha representa uma geração e em cada coluna o valor não descontado da cava final conforme figura abaixo:

EvolPit Versão 1.00

Parâmetros | Modelo | Histogramas | Resultados | Seções | Gr. Evol. | Evolução AG | Aptidão | Res. Grupo | Gr Multi

G/I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Med.
1	52	-8	-12	-8	0	-24	-24	32	-12	-32	20	-24	0	-24	8	-3.733
2	52	52	0	32	32	20	0	8	0	12	40	0	12	0	0	17.33
3	52	52	52	52	32	32	20	40	20	80	0	36	28	16	80	39.46
4	80	80	52	52	52	40	52	80	80	0	0	80	0	0	0	43.20
5	80	80	52	52	52	52	80	80	80	72	80	0	0	0	72	55.46
6	80	80	80	80	80	40	80	72	80	0	0	80	0	0	80	55.46
7	80	80	80	80	80	80	80	80	80	0	80	0	80	0	92	64.80
8	92	108	80	80	80	80	80	80	80	-60	72	80	0	0	0	56.79
9	108	92	52	80	80	80	80	80	80	72	80	108	0	0	0	66.13
10	108	60	80	80	80	80	80	80	72	80	108	80	0	0	0	65.86
11	108	80	80	80	80	80	80	92	80	108	80	80	0	0	0	68.53
12	108	80	80	80	80	80	80	92	80	108	96	0	80	0	0	69.59
13	108	40	80	80	80	80	80	92	80	60	96	108	80	80	80	81.59
14	108	48	96	108	80	80	80	92	80	108	96	92	0	0	0	71.19
15	108	108	96	80	80	-4	60	72	108	92	92	72	92	0	0	70.40
16	108	108	96	80	80	72	108	92	92	92	92	40	80	0	0	76
17	108	96	108	80	80	92	108	92	92	92	80	80	0	0	0	73.86
18	108	72	108	80	80	92	80	92	92	92	60	92	0	40	60	76.53
19	108	108	80	80	92	92	72	92	92	0	68	80	44	0	0	67.19
20	108	108	80	40	36	92	-4	92	92	68	64	0	-36	0	104	56.26
21	108	64	80	92	92	92	68	64	104	80	80	80	0	0	0	66.93
22	108	80	92	92	92	68	104	80	80	80	80	80	72	0	0	73.86
23	108	80	92	92	92	56	80	80	80	80	80	80	12	0	0	67.46

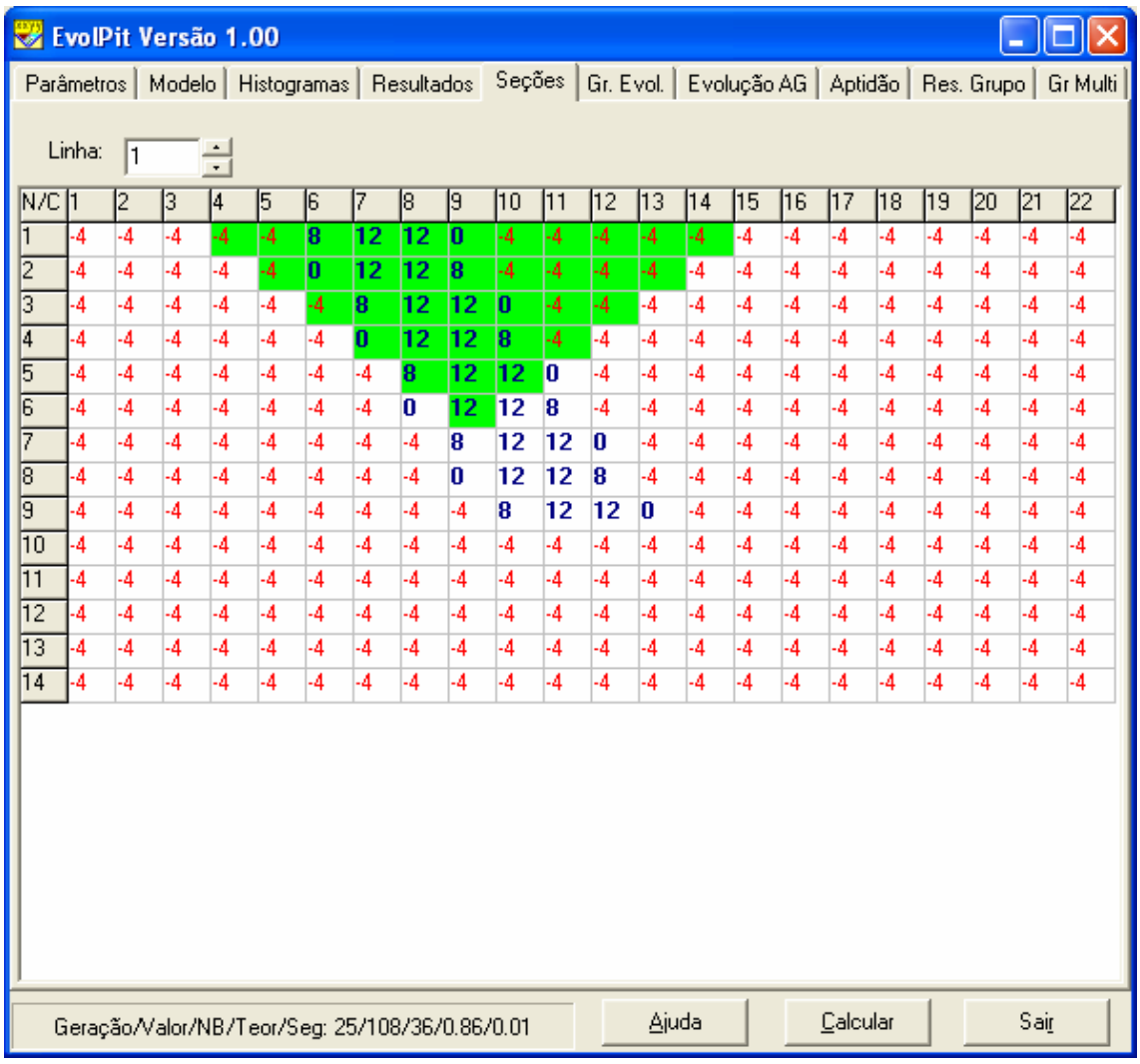
Geração/Valor/NB/Teor/Seg: 25/108/36/0.86/0.01 | Ajuda | Calcular | Sair

Figura 4.4.3.3.1 – Resultados de todos os indivíduos da população por geração (*Evolpit*).

Nessa interface é possível visualizar a evolução do algoritmo genético. Cada linha representa uma geração, e cada coluna um indivíduo. Por exemplo, o valor de -60 na linha 8 e coluna 10 significa que na oitava geração o décimo indivíduo da população obteve uma cava com um valor de menos sessenta unidades monetárias. Na primeira coluna, ter-se-á sempre o melhor indivíduo e, na última, o valor médio da população.

4.4.3.4 *Visualização dos blocos lavrados através de seções verticais*

A janela Seções permite a visualização de seções verticais do modelo de blocos, a partir da seleção da linha do modelo. Os blocos com valor negativo terão sempre a cor vermelha, e os de valores maiores ou igual a zero terão a cor azul. Os blocos contidos na cava final terão cor de fundo verde, conforme a figura abaixo:



N/C	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	-4	-4	-4	-4	-4	8	12	12	0	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
2	-4	-4	-4	-4	-4	0	12	12	8	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
3	-4	-4	-4	-4	-4	-4	8	12	12	0	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
4	-4	-4	-4	-4	-4	-4	0	12	12	8	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
5	-4	-4	-4	-4	-4	-4	-4	8	12	12	0	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
6	-4	-4	-4	-4	-4	-4	-4	0	12	12	8	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
7	-4	-4	-4	-4	-4	-4	-4	-4	8	12	12	0	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
8	-4	-4	-4	-4	-4	-4	-4	-4	0	12	12	8	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
9	-4	-4	-4	-4	-4	-4	-4	-4	-4	8	12	12	0	-4	-4	-4	-4	-4	-4	-4	-4	-4
10	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
11	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
12	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
13	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
14	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4

Figura 4.4.3.4.1 – Visualização dos blocos lavrados (*Evolpit*).

4.4.3.5 *Gráficos de evolução do AG*

Após o processamento dos cálculos existem duas maneiras de se visualizar a evolução do algoritmo genético implementado. Na primeira, existente na janela Gr.Evol. pode-se visualizar, a cada geração (eixo X), os valores encontrados (eixo Y) conforme se segue:

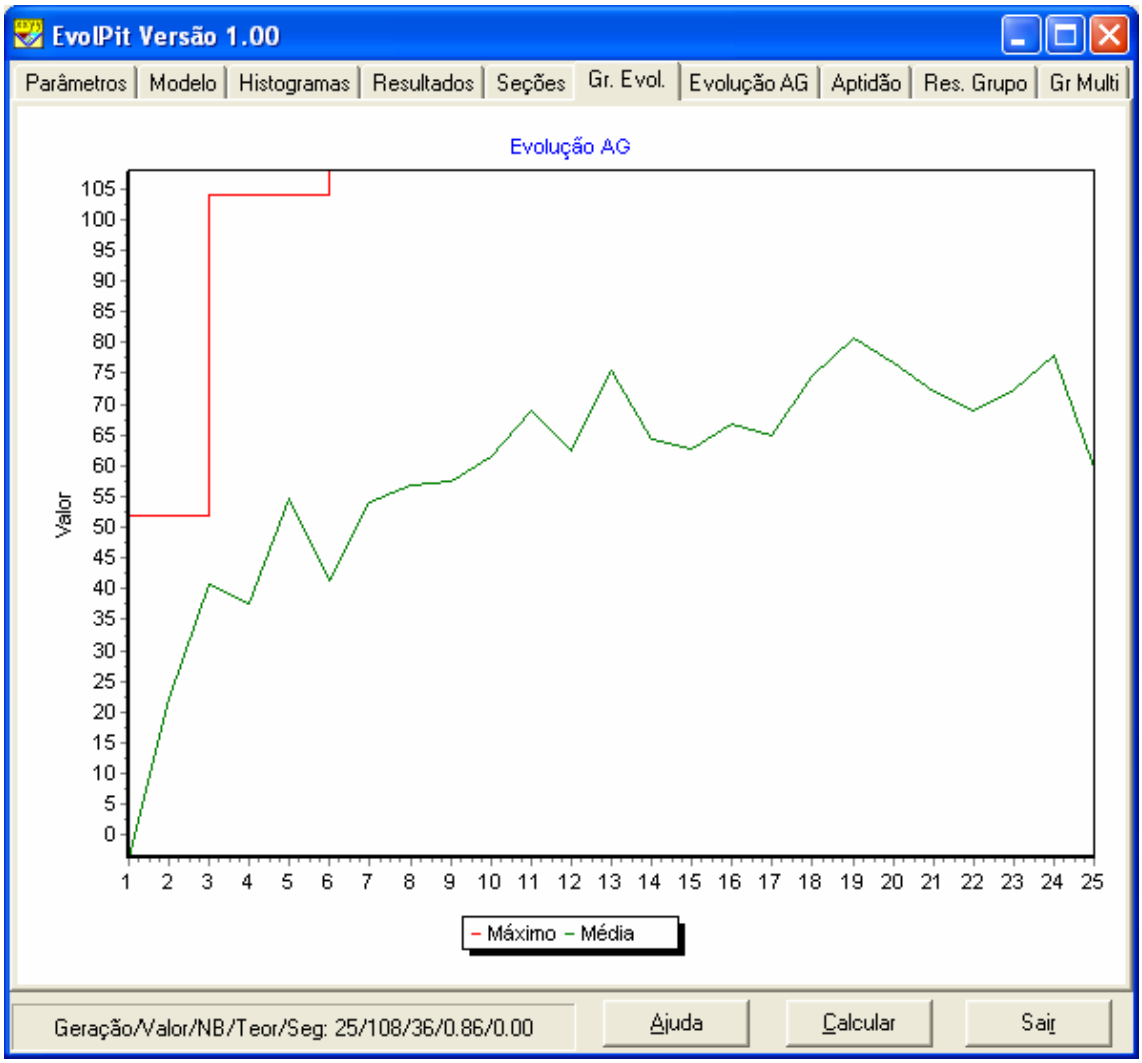


Figura 4.4.3.5.1 – Gráfico de evolução (*Evolpit*).

A segunda maneira de se visualizar a evolução do algoritmo é pela utilização dos recursos de cores por meio da combinação de um conjunto de seis barras verticais de corres e de uma barra horizontal que define a densidade de cores a ser utilizada pelo programa.

As figuras a seguir exibem em cada linha os mesmos parâmetros de entrada, porém com cores e densidades de cores diferentes:

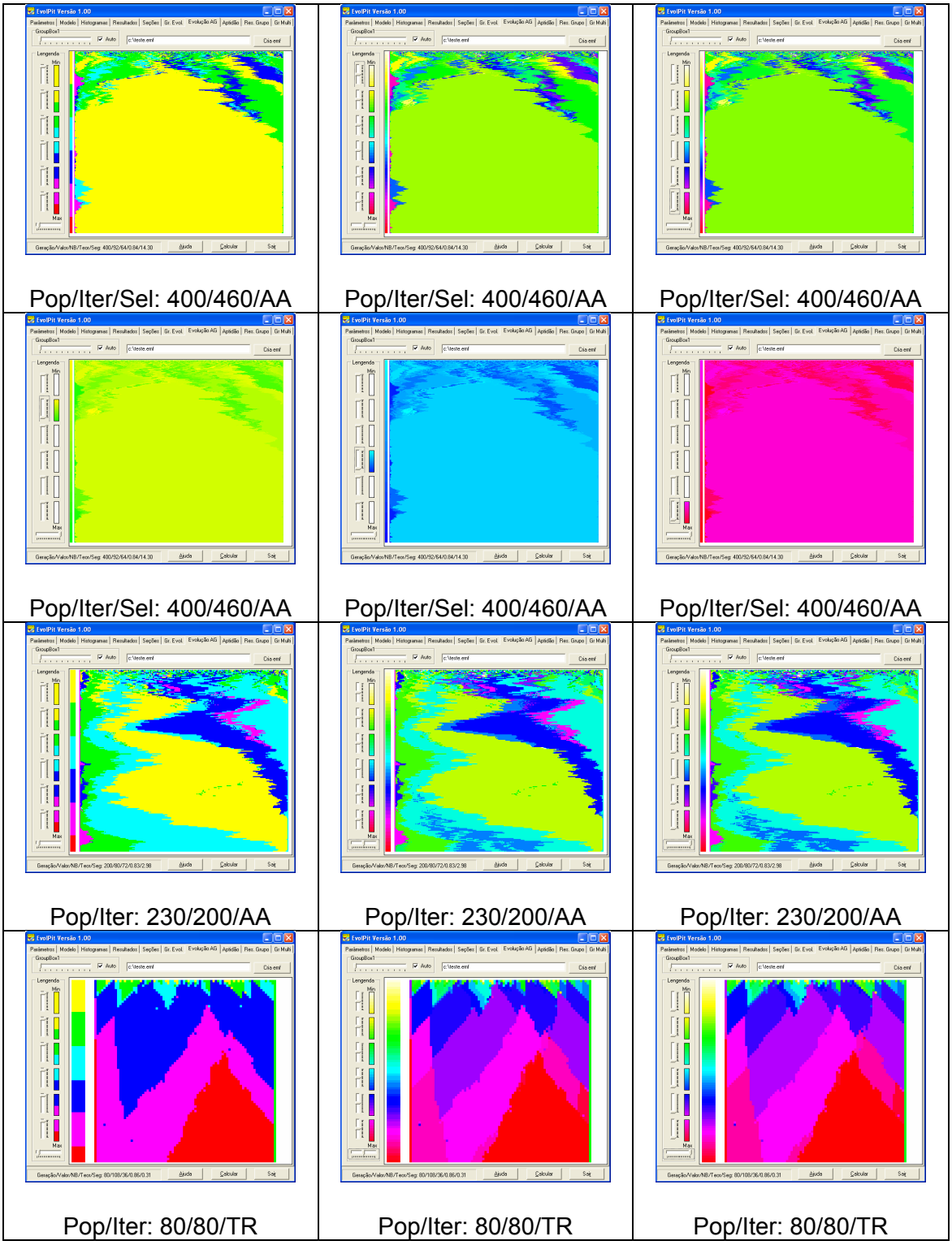
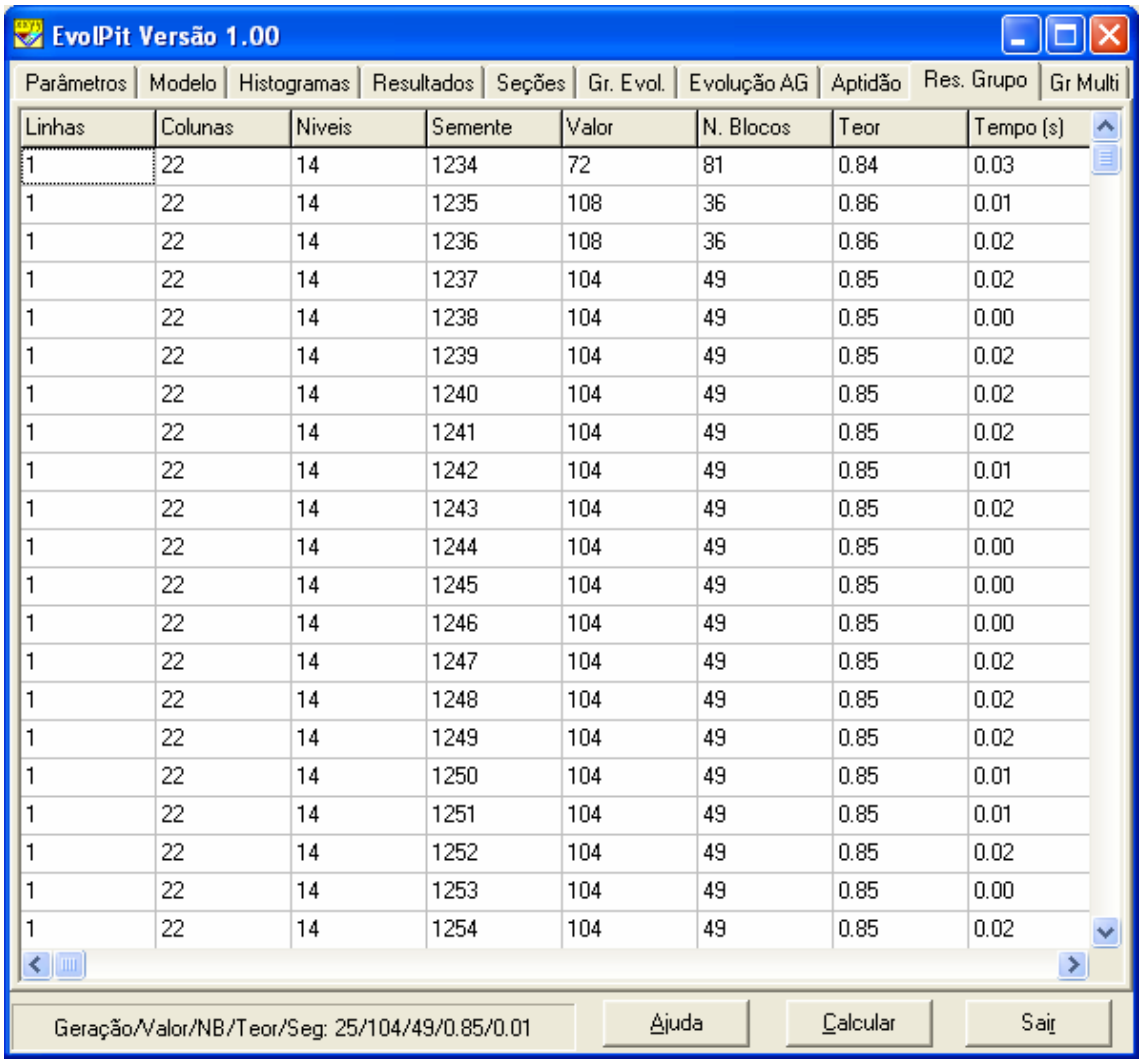


Figura 4.4.3.5.2 – Gráfico de evolução AG com diferentes resoluções (*Evolpit*).

4.4.3.6 *Resumo dos resultados para mais de um cenário de lavra*

Quando se define uma ou mais opções de *loop* nos parâmetros de semente, população, cruzamento, mutação e elitismo, pode-se visualizar um resumo dos resultados encontrados na janela Resultados, conforme abaixo:



Linhas	Colunas	Niveis	Semente	Valor	N. Blocos	Teor	Tempo (s)
1	22	14	1234	72	81	0.84	0.03
1	22	14	1235	108	36	0.86	0.01
1	22	14	1236	108	36	0.86	0.02
1	22	14	1237	104	49	0.85	0.02
1	22	14	1238	104	49	0.85	0.00
1	22	14	1239	104	49	0.85	0.02
1	22	14	1240	104	49	0.85	0.02
1	22	14	1241	104	49	0.85	0.02
1	22	14	1242	104	49	0.85	0.01
1	22	14	1243	104	49	0.85	0.02
1	22	14	1244	104	49	0.85	0.00
1	22	14	1245	104	49	0.85	0.00
1	22	14	1246	104	49	0.85	0.00
1	22	14	1247	104	49	0.85	0.02
1	22	14	1248	104	49	0.85	0.02
1	22	14	1249	104	49	0.85	0.02
1	22	14	1250	104	49	0.85	0.01
1	22	14	1251	104	49	0.85	0.01
1	22	14	1252	104	49	0.85	0.02
1	22	14	1253	104	49	0.85	0.00
1	22	14	1254	104	49	0.85	0.02

Figura 4.4.3.6.1 – Resumo dos resultados para mais de um cenário de lavra.

4.4.3.7 *Gráfico dos resultados da otimização multiobjetivo (Objetivo1 x Objetivo2)*

Os resultados, quando são utilizados os recursos de otimização multiobjetivo por meio das opções Valor_Minério, Valor_Teor e Minério_Teor, podem ser visualizados pelo o gráfico contido na janela da interface conforme a seguir:

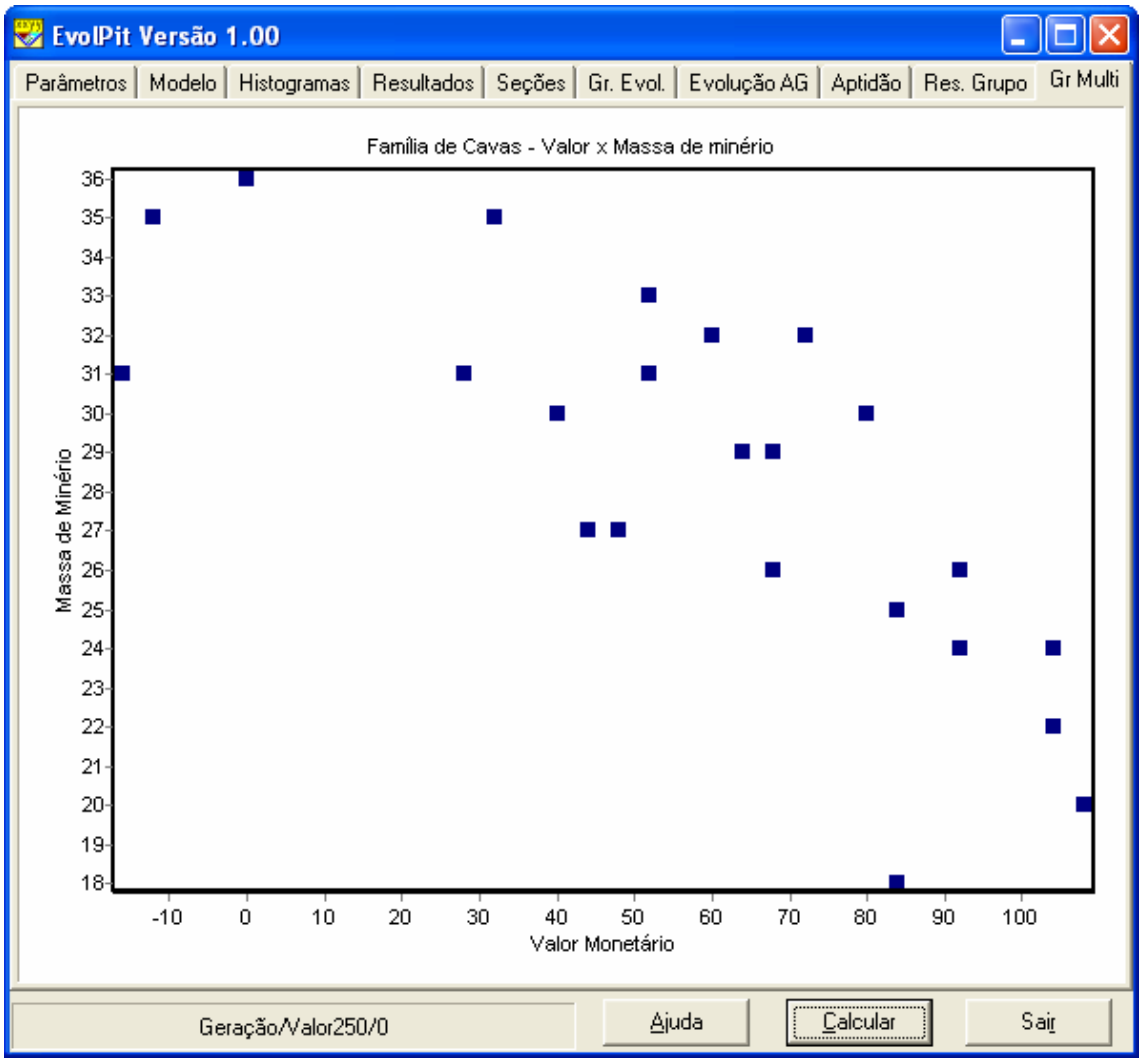


Figura 4.4.3.7.1 – Gráfico dos resultados da otimização multiobjetivo (Recurso X Valor).

5 Resultados dos algoritmos genéticos implementados

5.1 Testes e resultados encontrados AGS - *VBA*

No intuito de validar o funcionamento do algoritmo codificado e de analisar os diversos parâmetros de funcionamento do algoritmo genético ora implementado, foram realizadas inicialmente quatro baterias (25, 50, 100 e 200 gerações) de 32.400 simulações (**para uma seção**) com os diferentes parâmetros abaixo:

Número de gerações:	25; 50; 100 e 200
Tamanho da população:	10; 30; 50; 70 e 90
Probabilidade de cruzamento:	0,5; 0,6; 0,7; 0,8; 0,9 e 1,0
Probabilidade de mutação:	0; 0,02; 0,04; 0.06; 0,08 e 0,1
Probabilidade de elitismo:	0; 0,1; 0,2; 0,3; 0,4 e 0,5
Semente:	1234 a 1263 (trinta valores)

Para cada grupo de testes, foi calculado o percentual do ótimo que é a média do valor obtido dividido pela solução exata do problema. O gráfico a seguir ilustra a importância do tamanho da população para o problema proposto.

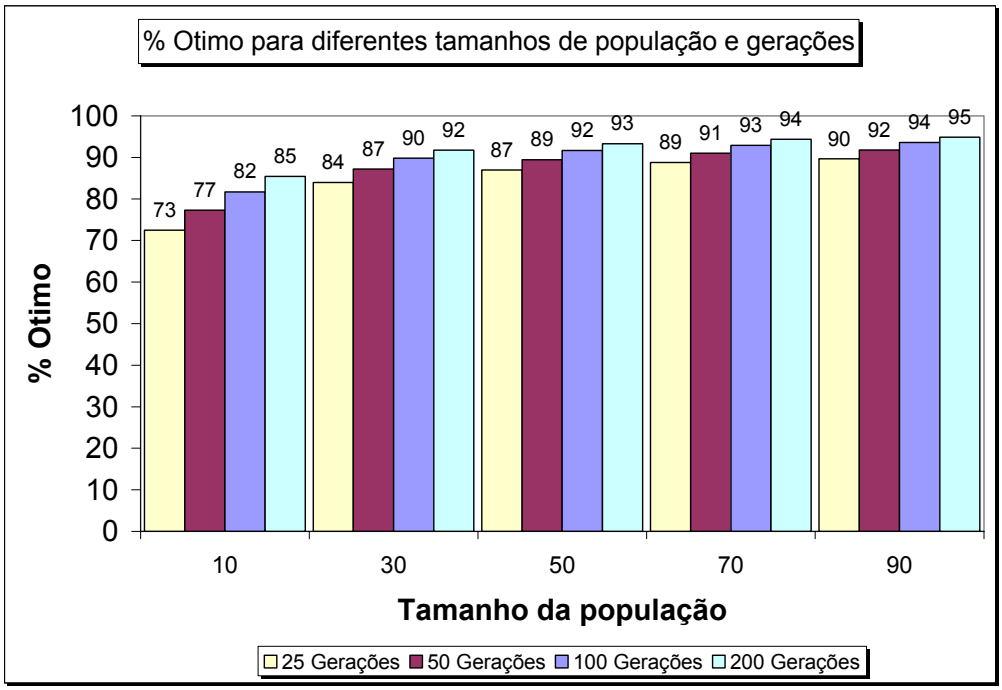


Figura 5.1.1 – Gráfico de % ótimo do tamanho da população (AGS – 1 Seção).

Conforme pode se observa, quanto maior o tamanho da população, maior a chance de o algoritmo convergir para a solução ótima.

O gráfico abaixo exibe a variação do percentual de acerto para as diversas combinações de número de gerações utilizadas:

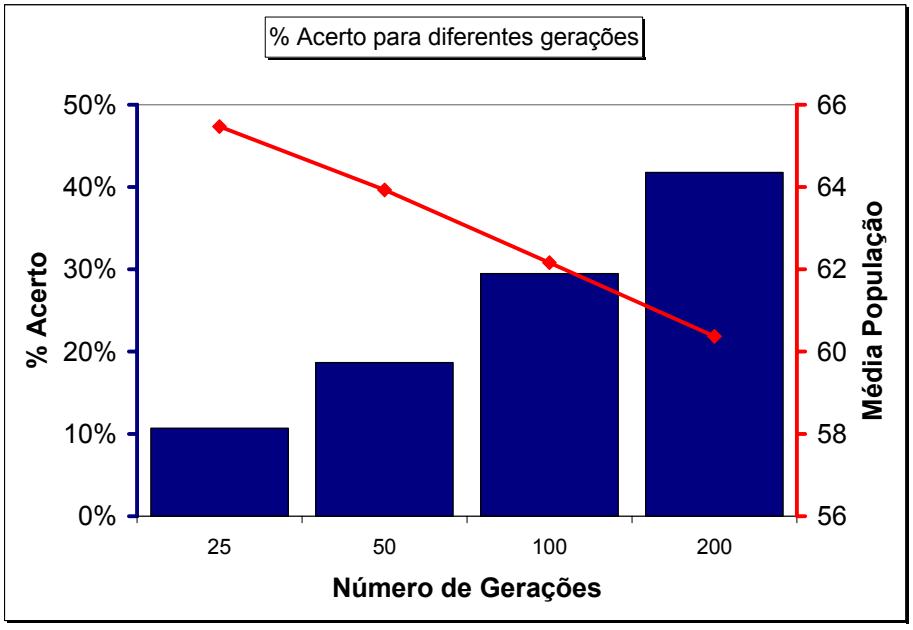


Figura 5.1.2 – Gráfico de % acerto do número de gerações (AGS – 1 Seção).

Observa-se uma relação direta do número de gerações com o percentual de acerto, ou seja, quanto maior o número de gerações, maior o percentual de sucesso. Nota-se que, com o aumento do número de gerações, menor é a média da população que convergiu para a solução correta.

O gráfico a seguir ilustra o comportamento da probabilidade de cruzamento para o problema proposto:

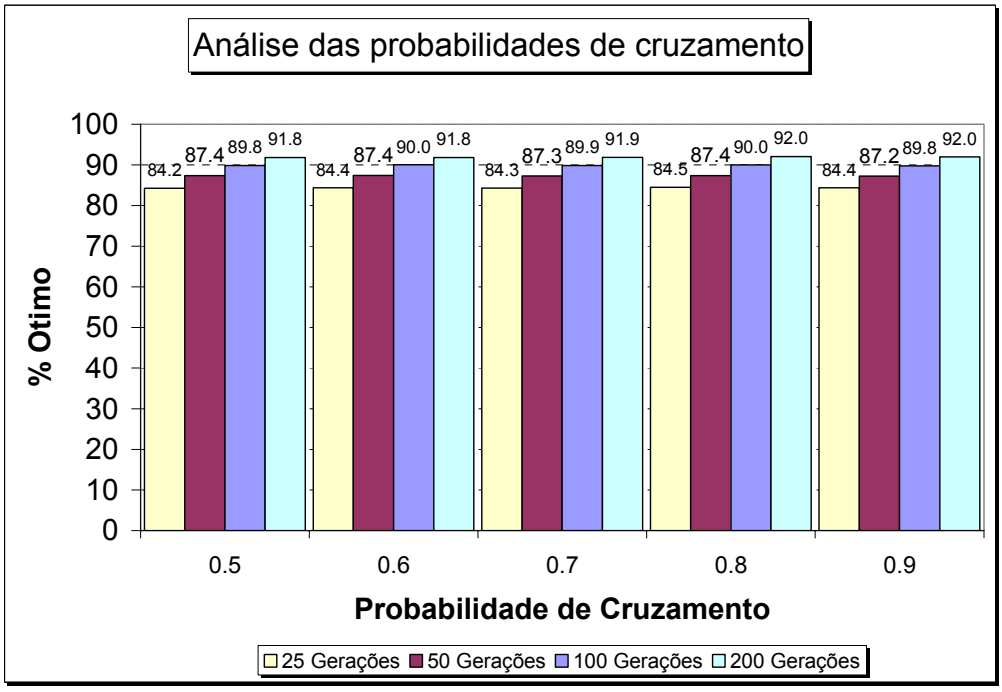


Figura 5.1.3 – Gráfico de % ótimo da probabilidade de cruzamento (AGS – 1 Seção).

Percebe-se que, para a faixa de valores de probabilidade de cruzamento utilizados no problema proposto, não houve diferenças significativas.

O gráfico a seguir ilustra o comportamento da probabilidade de mutação para o problema proposto:

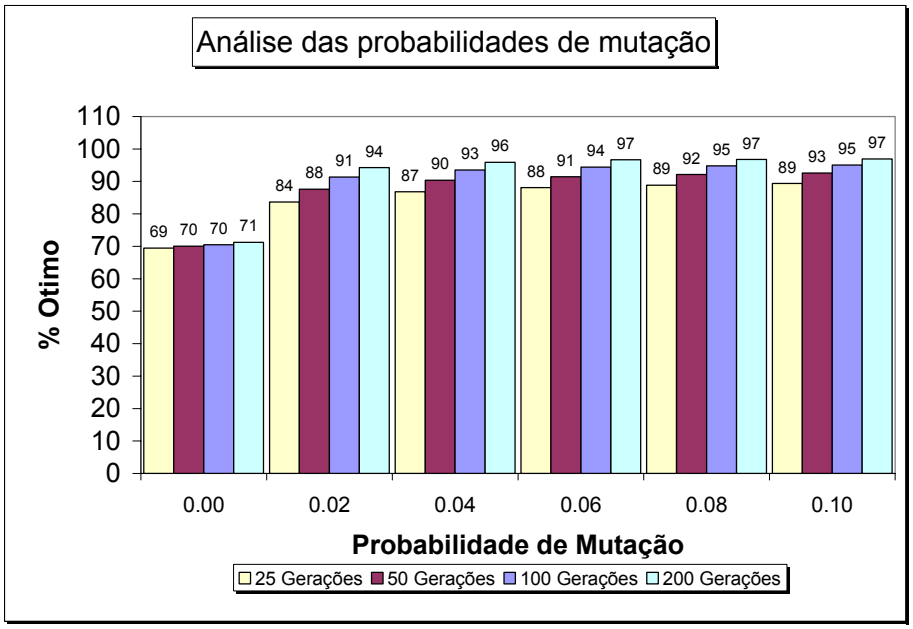


Figura 5.1.4 – Gráfico de % ótimo da probabilidade de mutação (AGS – 1 Seção).

Conforme exibido no gráfico, houve aumento crescente na convergência do algoritmo, de acordo com o aumento da probabilidade de mutação, principalmente para a faixa inicial dos valores simulados.

O gráfico a seguir ilustra o comportamento da probabilidade de elitismo para o problema proposto.

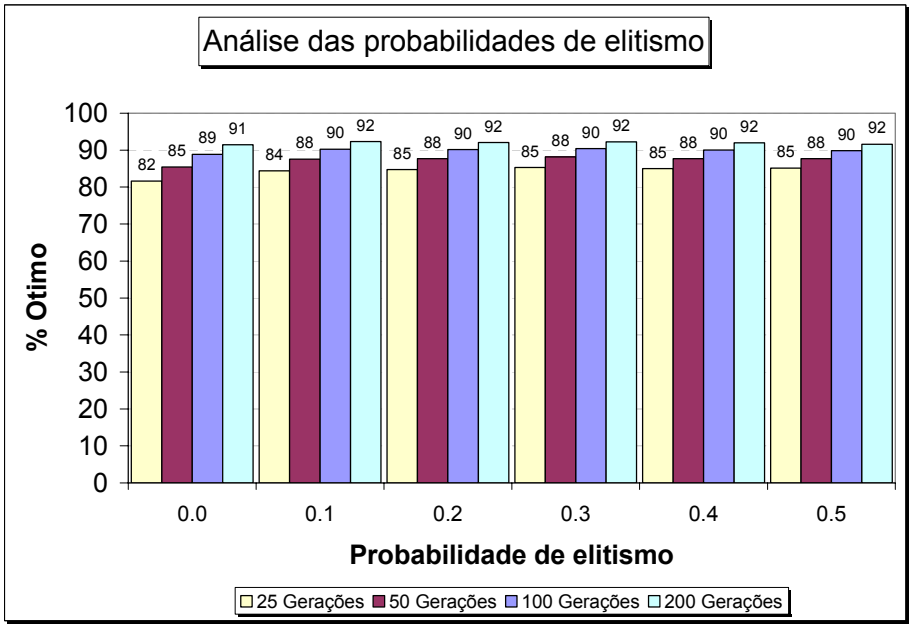


Figura 5.1.5 – Gráfico de % ótimo da probabilidade de elitismo (AGS – 1 Seção).

Observa-se que, mesmo com o aumento do número de cópias do melhor indivíduo nas diversas gerações, o algoritmo permaneceu praticamente constante, para as diversas taxas de elitismo simuladas.

Como para uma seção o algoritmo conseguiu convergir para a solução ótima, foi realizado mais um conjunto de simulações, com a mesma combinação de parâmetros utilizados anteriormente, para duas seções análogas à primeira.

O gráfico abaixo ilustra a importância do tamanho da população para o problema proposto (duas seções).

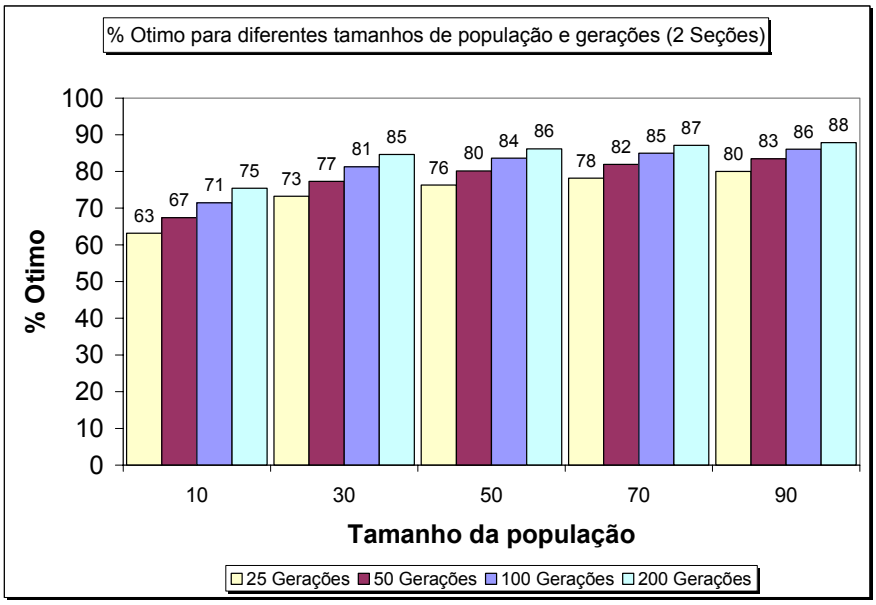


Figura 5.1.6 – Gráfico de % ótimo do tamanho da população (AGS – 2 Seções).

Conforme pode se observa, quanto maior o tamanho da população, maior a chance de o algoritmo convergir para a solução ótima. Percebe-se, ainda, que o percentual do ótimo é inferior para a população de noventa indivíduos (88%) quando comparado ao obtido para uma seção (95%).

Um terceiro teste foi realizado considerando o mesmo problema para três seções, com a mesma combinação de parâmetros, porém somente com 100 gerações, ou seja, mais 32.400 simulações. Nesse teste, obteve-se somente um resultado positivo, ou seja, apenas uma simulação conseguiu convergir para a solução correta. A combinação de parâmetros foi de 0,5 de probabilidade de cruzamento, 0,06 de mutação, e 0,5 de elitismo. Obtiveram-se, ainda, dois resultados com valor de 320 o

que equivale a 98,8% da solução ótima e 482 valores com valor de 312 correspondente a 96,3% do valor da solução ótima.

5.1.1 Conclusões dos resultados do algoritmo genético simples com elitismo *VBA*

O algoritmo genético simples (AGS) com elitismo implementado mostrou-se eficaz na resolução do problema para uma seção. Quando o número de seções aumenta há uma demanda computacional maior e maior risco de o algoritmo não convergir para a solução ótima.

Percebe-se que o tamanho da população e o número de gerações influem fortemente nos resultados, ou seja, com populações e gerações menores, o risco de o algoritmo convergir para máximos locais é maior.

Com relação à identificação dos melhores parâmetros de probabilidade de cruzamento, mutação e elitismo, para os problemas formulados para um tamanho fixo do número de gerações igual a duzentos e para uma seção, obteve-se *Pc* = 0,5 e 0,8, *Pm* = 0,06 e 0,08 e *Pe* = 0,2, considerando o percentual de sucesso das simulações realizadas para uma seção. Para o problema de duas seções, os melhores parâmetros foram *Pc* = 0,7; *Pm* = 0,06 e 0,08 e *Pe* = 0,5.

5.2 Testes e resultados encontrados problema mono-objetivo - Evolpit - *Delphi*

Analogamente aos testes realizados no algoritmo codificado em *VBA*, foram analisados os diversos parâmetros de funcionamento do algoritmo genético codificado em *Delphi*. Foram realizadas inicialmente dez baterias (todos os métodos de seleção com 25 gerações) totalizando 32.400 simulações/método (para uma seção) conforme os diferentes parâmetros abaixo:

Método de seleção:	RO; AA; AE; AD; D1; D2; D3; TR; T1 e T2
Tamanho da população:	10; 30; 50; 70 e 90

Probabilidade de cruzamento:	0,5; 0,6; 0,7; 0,8; 0,9 e 1,0
Probabilidade de mutação:	0; 0,02; 0,04; 0.06; 0,08 e 0,1
Percentual de elitismo:	0; 0,1; 0,2; 0,3; 0,4 e 0,5
Semente:	1234 a 1263 (trinta valores)

Para cada grupo de testes, foi calculado o percentual de acerto, que é o número de sucessos obtidos dividido pelo número de testes realizados em percentagem. O gráfico abaixo ilustra os resultados obtidos para os diversos tipos de seleção codificados.

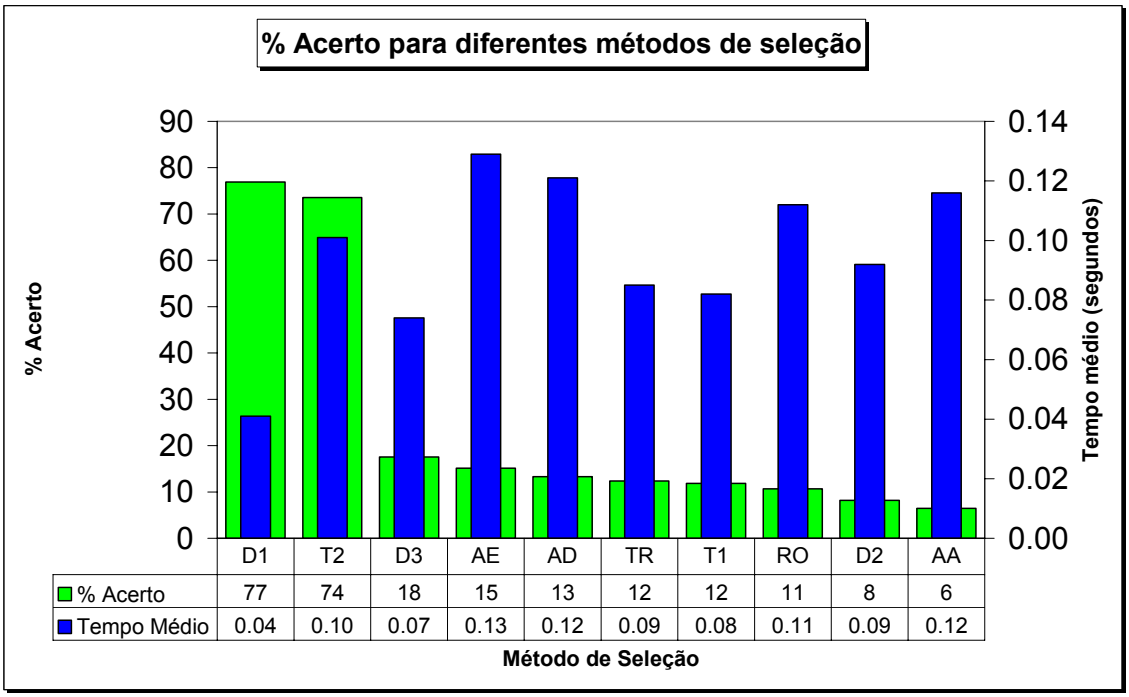
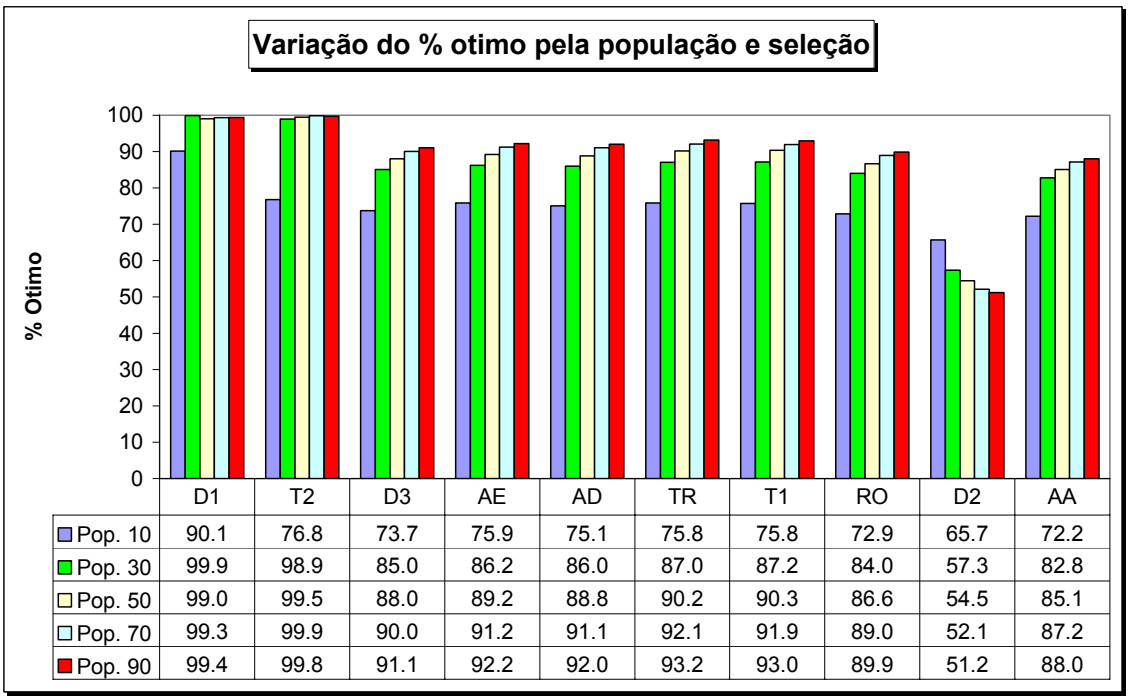


Figura 5.2.1 – Gráfico de % acerto dos métodos de seleção (*Evolpit* – 1 Seção).

Nesse gráfico, os resultados foram ordenados de acordo com o % de acerto para cada método de seleção implementado. Conforme se observa, o método de seleção de amostragem determinística variante 1 foi o mais eficiente pois obteve maior percentual de acerto (77%) e menor tempo de médio processamento (0.04 segundos). Como era esperado, o pior método foi o de amostragem aleatória, pois nele não é considerada a aptidão do indivíduo. Nota-se que existem 7 métodos de seleção superiores ao método da roleta que foi codificado em *VBA*.

O gráfico a seguir apresenta a influência do tamanho da população nos diferentes tipos de seleção codificados:

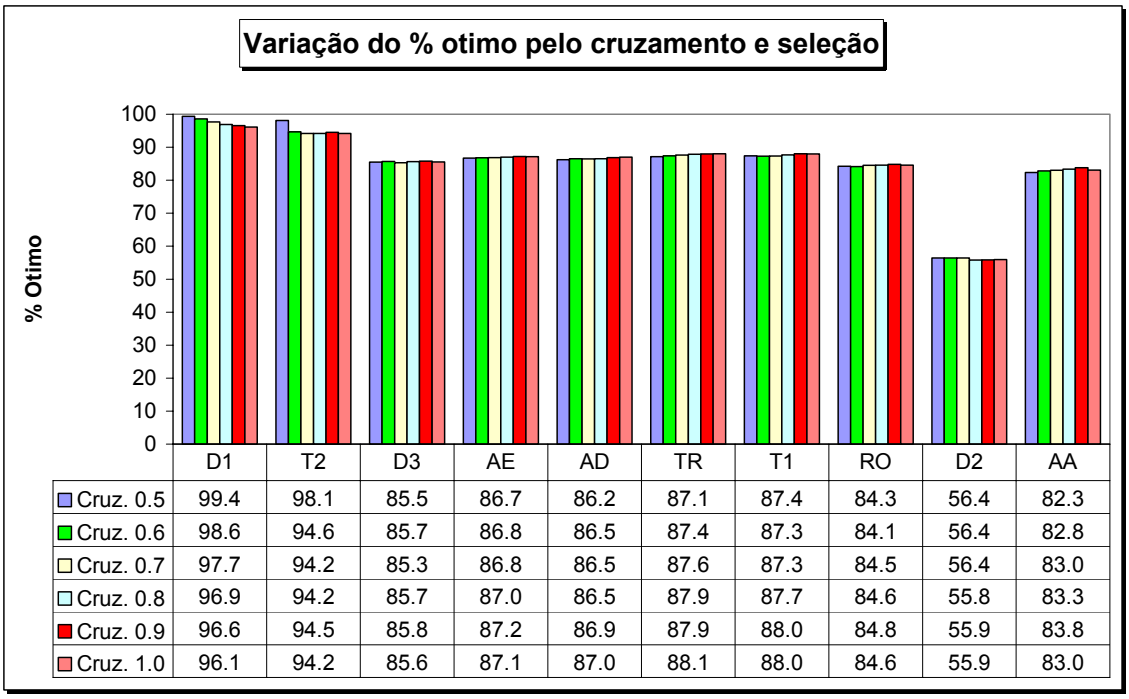


	D1	T2	D3	AE	AD	TR	T1	RO	D2	AA
Pop. 10	90.1	76.8	73.7	75.9	75.1	75.8	75.8	72.9	65.7	72.2
Pop. 30	99.9	98.9	85.0	86.2	86.0	87.0	87.2	84.0	57.3	82.8
Pop. 50	99.0	99.5	88.0	89.2	88.8	90.2	90.3	86.6	54.5	85.1
Pop. 70	99.3	99.9	90.0	91.2	91.1	92.1	91.9	89.0	52.1	87.2
Pop. 90	99.4	99.8	91.1	92.2	92.0	93.2	93.0	89.9	51.2	88.0

Figura 5.2.2 – Variação do % ótimo pela população e seleção (*Evolpit* – 1 seção).

Somente para o método de amostragem determinística variante 2 (D2), o aumento da população não incidiu sobre o percentual de ótimo. Sendo que, para o método de amostragem determinística variante 1, o tamanho que obteve maior percentual do ótimo foi o de 30 indivíduos (99,9%). Já o método de torneio variante 2 foi para o tamanho da população igual a 70, o que obteve o maior percentual do ótimo (99.9%).

O gráfico a seguir apresenta a influência da probabilidade de cruzamento nos diferentes tipos de seleção:

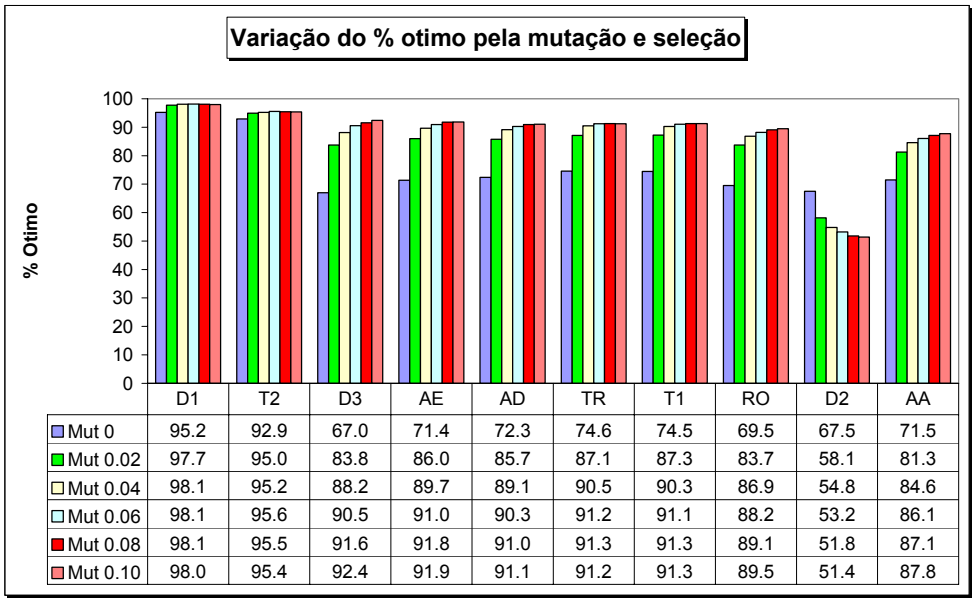


	D1	T2	D3	AE	AD	TR	T1	RO	D2	AA
Cruz. 0.5	99.4	98.1	85.5	86.7	86.2	87.1	87.4	84.3	56.4	82.3
Cruz. 0.6	98.6	94.6	85.7	86.8	86.5	87.4	87.3	84.1	56.4	82.8
Cruz. 0.7	97.7	94.2	85.3	86.8	86.5	87.6	87.3	84.5	56.4	83.0
Cruz. 0.8	96.9	94.2	85.7	87.0	86.5	87.9	87.7	84.6	55.8	83.3
Cruz. 0.9	96.6	94.5	85.8	87.2	86.9	87.9	88.0	84.8	55.9	83.8
Cruz. 1.0	96.1	94.2	85.6	87.1	87.0	88.1	88.0	84.6	55.9	83.0

Figura 5.2.3 – Variação do % ótimo pelo cruzamento e seleção (*Evolpit* – 1 seção).

Conforme se observa no gráfico 5.2.3, existe uma tendência de que, quanto menor for a taxa de cruzamento, maior a probabilidade de acerto para os dois primeiros métodos (D1 e T2). Para os demais métodos de seleção, as melhores taxas de cruzamento variam de método para método. Por exemplo, para o método de amostragem determinística, a melhor taxa de cruzamento foi de 0.9. Já para o método de amostragem aleatória, a taxa de 0.5 foi a pior.

A influência da probabilidade de mutação nos diferentes tipos de seleção é apresentada no gráfico a seguir:

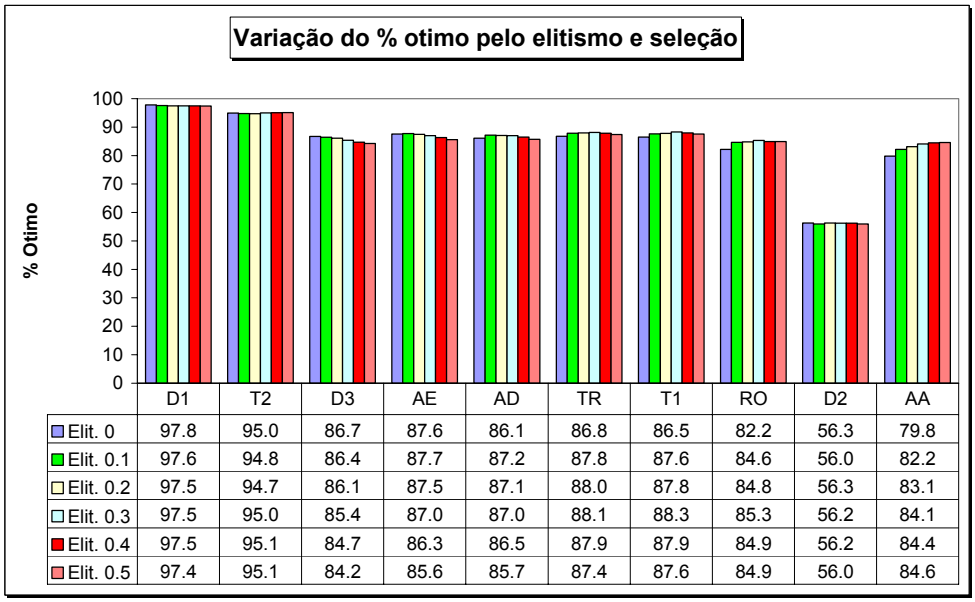


	D1	T2	D3	AE	AD	TR	T1	RO	D2	AA
Mut 0	95.2	92.9	67.0	71.4	72.3	74.6	74.5	69.5	67.5	71.5
Mut 0.02	97.7	95.0	83.8	86.0	85.7	87.1	87.3	83.7	58.1	81.3
Mut 0.04	98.1	95.2	88.2	89.7	89.1	90.5	90.3	86.9	54.8	84.6
Mut 0.06	98.1	95.6	90.5	91.0	90.3	91.2	91.1	88.2	53.2	86.1
Mut 0.08	98.1	95.5	91.6	91.8	91.0	91.3	91.3	89.1	51.8	87.1
Mut 0.10	98.0	95.4	92.4	91.9	91.1	91.2	91.3	89.5	51.4	87.8

Figura 5.2.4 – Variação do % ótimo pela mutação e seleção (*Evolpit* – 1 seção).

Observa-se que, mesmo na ausência de mutação, o algoritmo obteve um alto percentual do ótimo, principalmente nos dois primeiros métodos (D1 e T2). Para os outros métodos de seleção, houve a tendência de que, com o aumento da taxa de mutação, maior a probabilidade de o algoritmo convergir para a solução ótima.

O gráfico a seguir apresenta a influência do percentual de elitismo nos diferentes tipos de seleção codificados:

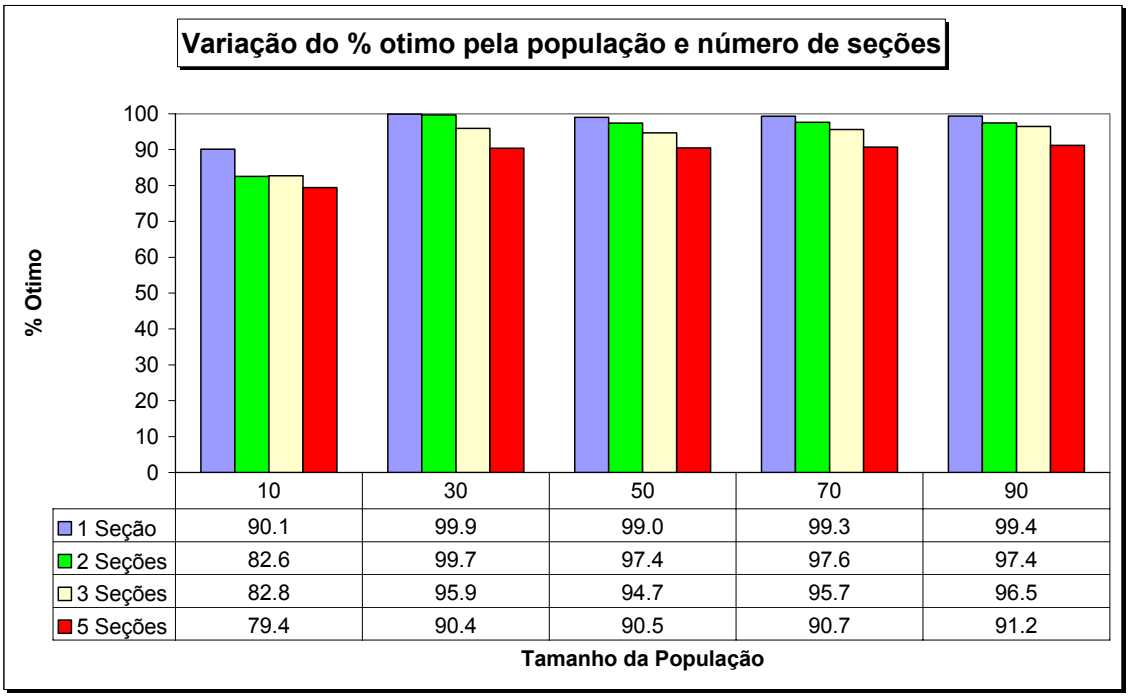


	D1	T2	D3	AE	AD	TR	T1	RO	D2	AA
Elit. 0	97.8	95.0	86.7	87.6	86.1	86.8	86.5	82.2	56.3	79.8
Elit. 0.1	97.6	94.8	86.4	87.7	87.2	87.8	87.6	84.6	56.0	82.2
Elit. 0.2	97.5	94.7	86.1	87.5	87.1	88.0	87.8	84.8	56.3	83.1
Elit. 0.3	97.5	95.0	85.4	87.0	87.0	88.1	88.3	85.3	56.2	84.1
Elit. 0.4	97.5	95.1	84.7	86.3	86.5	87.9	87.9	84.9	56.2	84.4
Elit. 0.5	97.4	95.1	84.2	85.6	85.7	87.4	87.6	84.9	56.0	84.6

Figura 5.2.5 – Variação do % ótimo pelo elitismo e seleção (*Evolpit* – 1 seção)

Mesmo com a variação do percentual de elitismo, não houve uma tendência de convergência para o ótimo quando se comparam todos os métodos de seleção.

Como os melhores resultados para uma seção foram obtidos pelo método de amostragem determinística variante 1, foram realizadas mais três baterias de 32.400 simulações para duas, três e cinco seções. O gráfico abaixo ilustra o impacto do tamanho da população nos resultados:

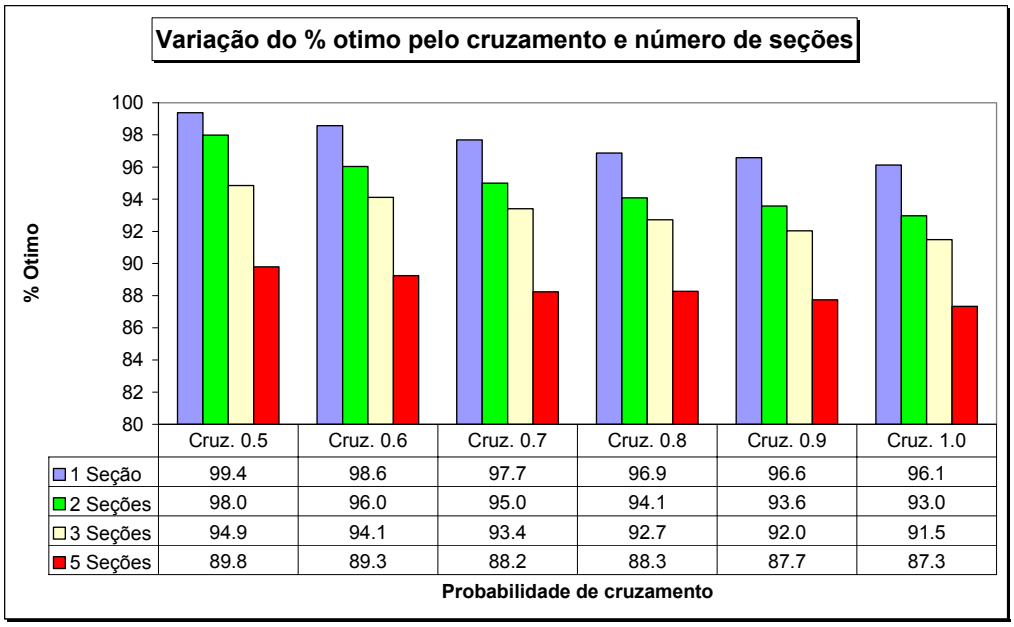


	10	30	50	70	90
1 Seção	90.1	99.9	99.0	99.3	99.4
2 Seções	82.6	99.7	97.4	97.6	97.4
3 Seções	82.8	95.9	94.7	95.7	96.5
5 Seções	79.4	90.4	90.5	90.7	91.2

Figura 5.2.6 – Variação do % ótimo pela população e número de seções (*Evolpit*).

Conforme se observa, para os modelos de recursos contendo uma e duas seções verticais, os melhores resultados foram para a população de 30 indivíduos. Já para os modelos de 3 e 5 seções, quanto maior o tamanho da população, maior a probabilidade de o algoritmo convergir para a solução correta.

O gráfico a seguir apresenta a influência da probabilidade de cruzamento nos diferentes modelos de recursos minerais:

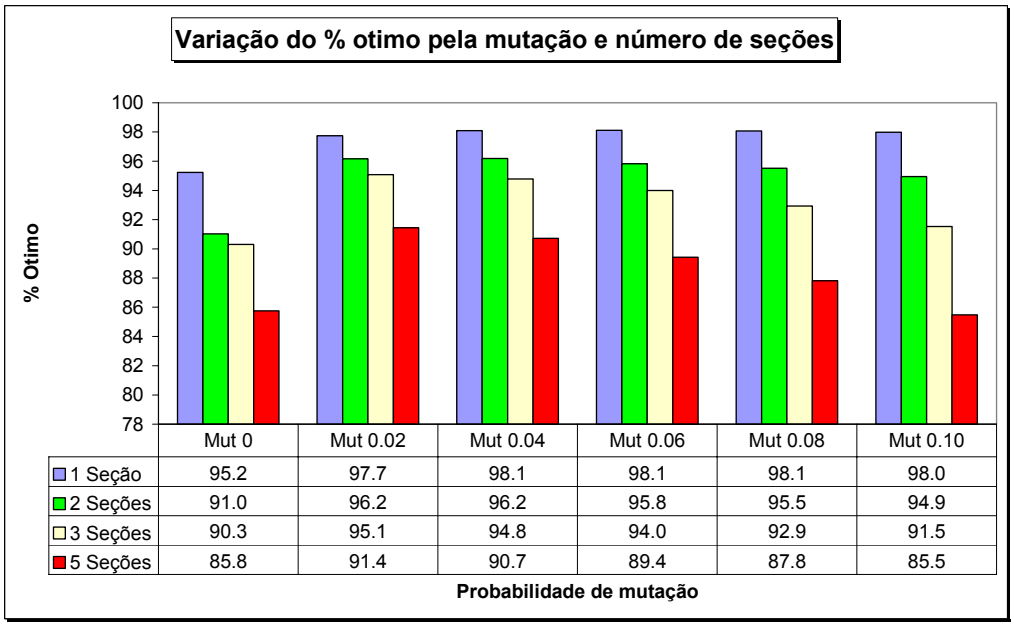


	Cruz. 0.5	Cruz. 0.6	Cruz. 0.7	Cruz. 0.8	Cruz. 0.9	Cruz. 1.0
1 Seção	99.4	98.6	97.7	96.9	96.6	96.1
2 Seções	98.0	96.0	95.0	94.1	93.6	93.0
3 Seções	94.9	94.1	93.4	92.7	92.0	91.5
5 Seções	89.8	89.3	88.2	88.3	87.7	87.3

Figura 5.2.7 – Variação do % ótimo pelo cruzamento e número de seções (*Evolpit*).

Em geral, quanto menor a taxa de cruzamento, maior o percentual do ótimo.

O gráfico a seguir apresenta a variação do percentual do ótimo em função da probabilidade de mutação para os 4 modelos de recursos construídos.

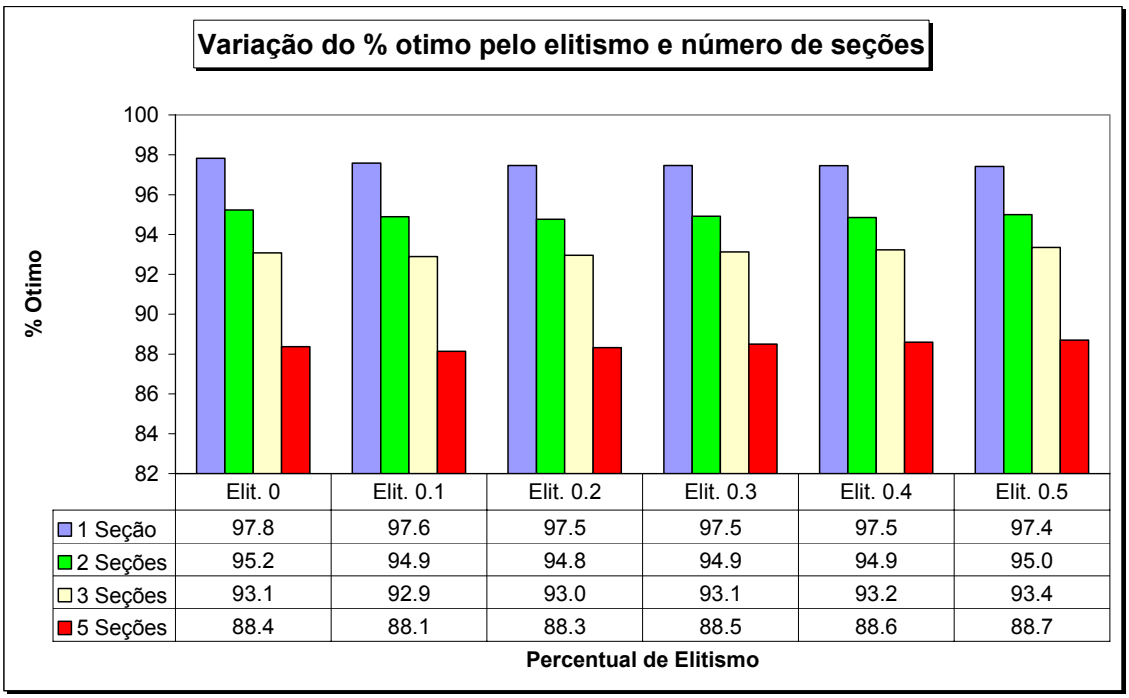


	Mut 0	Mut 0.02	Mut 0.04	Mut 0.06	Mut 0.08	Mut 0.10
1 Seção	95.2	97.7	98.1	98.1	98.1	98.0
2 Seções	91.0	96.2	96.2	95.8	95.5	94.9
3 Seções	90.3	95.1	94.8	94.0	92.9	91.5
5 Seções	85.8	91.4	90.7	89.4	87.8	85.5

Figura 5.2.8 – Variação do % ótimo pela mutação e número de seções (*Evolpit*).

Percebe-se que, com o aumento do número de blocos no modelo de recursos, por meio do incremento do número de seções, a taxa de mutação de 0.02 foi a que obteve melhores resultados.

A influência do percentual de elitismo, nos diferentes modelos de recursos, é apresentada no gráfico a seguir:



	Elit. 0	Elit. 0.1	Elit. 0.2	Elit. 0.3	Elit. 0.4	Elit. 0.5
1 Seção	97.8	97.6	97.5	97.5	97.5	97.4
2 Seções	95.2	94.9	94.8	94.9	94.9	95.0
3 Seções	93.1	92.9	93.0	93.1	93.2	93.4
5 Seções	88.4	88.1	88.3	88.5	88.6	88.7

Figura 5.2.9 – Variação do % ótimo pelo elitismo e número de seções (*Evolpit*).

Observa-se que não houve variações na % do ótimo para os diversos percentuais de elitismo analisados.

De acordo com os melhores resultados obtidos para o modelo de recursos de 5 seções verticais, foram simulados mais 3.600 parâmetros, visando aumentar as chances de o algoritmo convergir para a melhor solução, conforme abaixo:

Método de seleção: D1

Tamanho da população: 90

Probabilidade de cruzamento: 0,7

Probabilidade de mutação: 0.001 a 0.02 (20 incrementos)

Percentual de elitismo: 0,5 a 1,0 (6 incrementos)

Semente: 1234 a 1263 (trinta valores)

Para o número de gerações igual a 25, não houve nenhum resultado que convergiu para a solução correta do problema nas 3.600 simulações (5 seções). O percentual médio da solução ótima foi de 68%. Já para um número de gerações de 150 utilizando com *PCruz* = 0.7; *PMut* = 0.02 e *PElit* = 0.3 com 30 valores de semente, obtiveram-se 28 resultados corretos em 30, o que equivale a 93% de acerto. Utilizando-se os mesmos parâmetros com 50 gerações, obtiveram-se 14 acertos, ou seja, 47% de acerto.

Foi criado um modelo de recursos com 10 seções verticais e utilizando 150 gerações com uma população de 150 indivíduos, obtiveram-se 4 resultados corretos em 30 (sementes). Para um modelo de 100 seções, com os seguintes parâmetros tem-se:

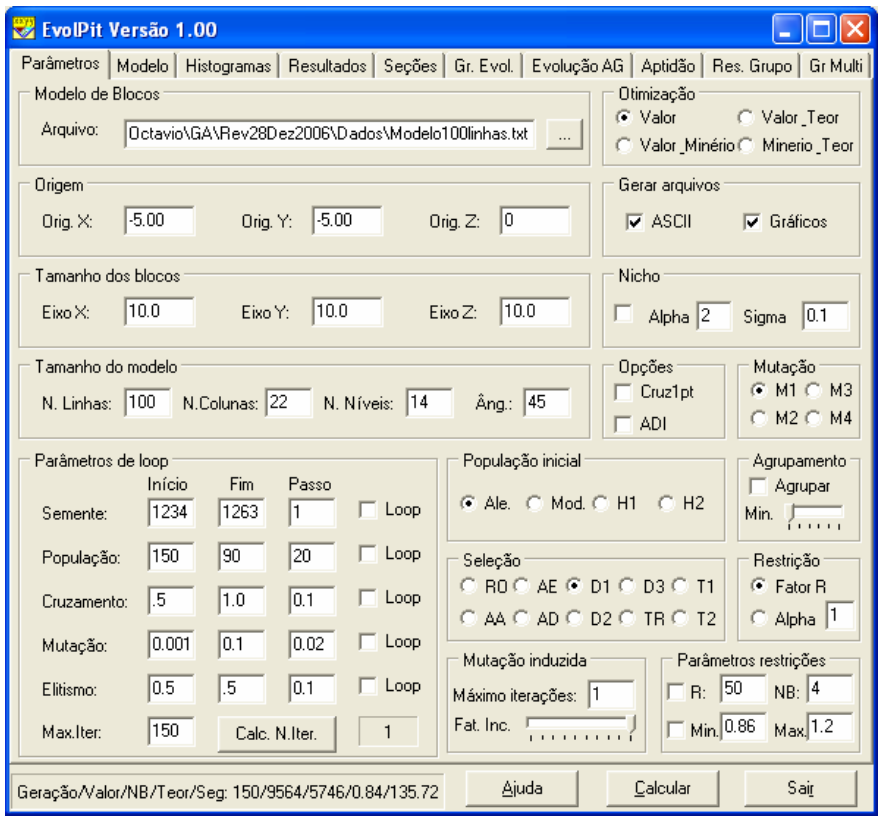
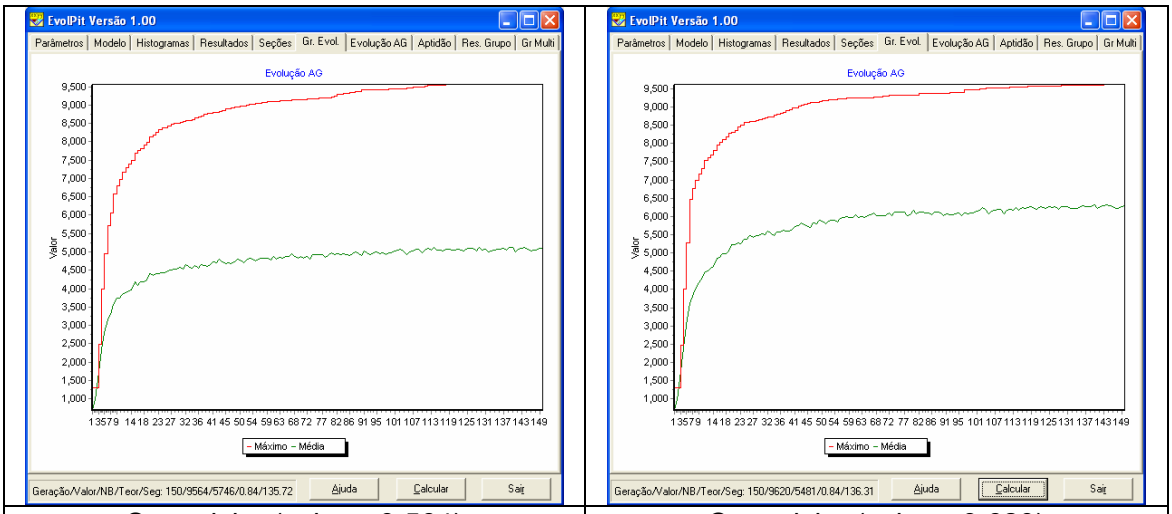
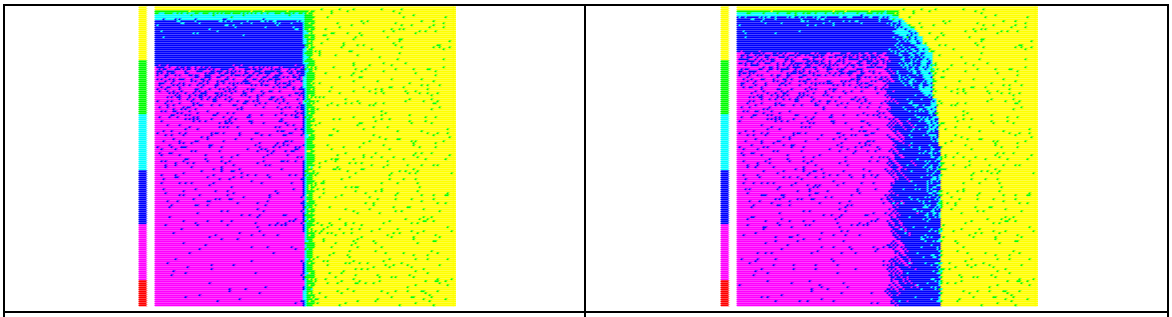


Figura 5.2.10 – Parâmetros de entrada 100 seções (*Evolpit*).



Sem nicho (valor = 9.564)	Com nicho (valor = 9.620)

Figura 5.2.11 – Gráfico de evolução 100 seções (Evolpit).

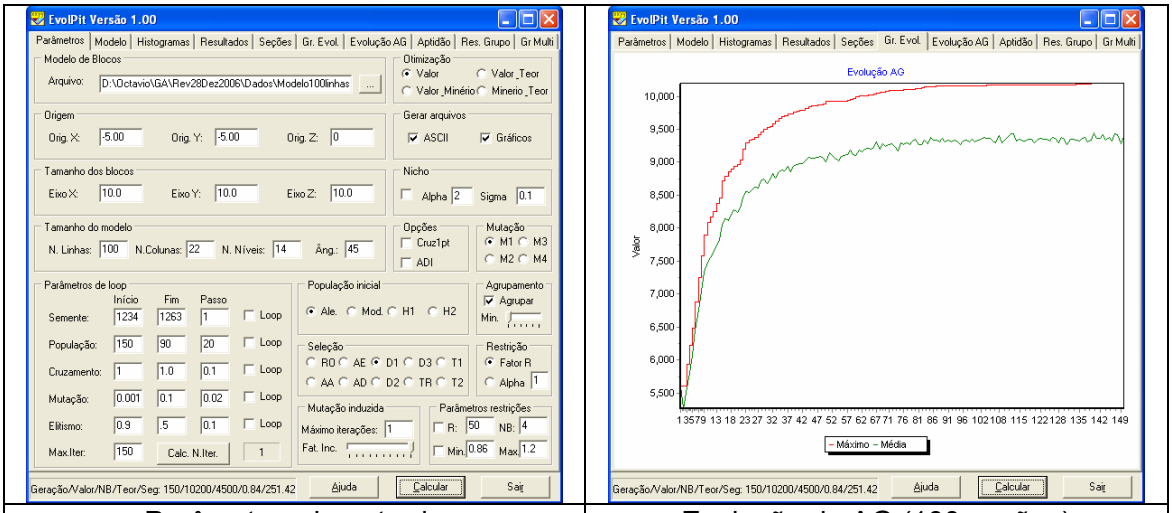


Sem nicho	Com nicho

Figura 5.2.12 – Gráfico de evolução por indivíduo 100 seções (*Evolpit*).

Conforme pode se observa nas figuras 5.2.11 e 5.2.12, houve uma pequena melhora nos resultados para o modelo de recursos contendo 100 seções verticais, utilizando-se os recursos de nicho do algoritmo (de 9.564 para 9.620 unidades monetárias), o que corresponde a 89% da solução ótima.

Utilizando-se os recursos de agrupamento do algoritmo, conforme descrito no item 4.4.2.4, após vários testes com os diversos parâmetros do algoritmo, foram obtidos os seguintes resultados:



Parâmetros de entrada	Evolução do AG (100 seções)

Figura 5.2.13 – Parâmetros e evolução para 100 seções com agrupamento (*Evolpit*).

Conforme pode se observa, o algoritmo conseguiu convergir para 94,4% (10200/10800) da solução ótima, para um modelo de recursos de 100 seções verticais (36.000 positivos).

5.2.1 Conclusões dos resultados do problema mono-objetivo - Evolpit – *Delphi*

Quando comparados os resultados obtidos com o algoritmo genético simples AGS com elitismo, percebe-se que o *Evolpit* obteve um desempenho melhor, tanto em termos de convergência para a solução ótima, quanto em tempo de processamento. Tal diferença pode ser atribuída aos métodos de seleção que foram mais eficientes para o problema proposto, bem como à utilização do compilador *Delphi*.

O algoritmo implementado no programa *Evolpit* foi capaz de gerar soluções eficientes com um relativo baixo custo computacional para o modelo de blocos de uma, duas e dez seções. Porém, quando o número de seções aumenta, o mesmo possui um custo computacional elevado, além de não conseguir convergir para a solução ótima, como para o exemplo do modelo de recursos contendo 100 seções.

5.3 Testes e resultados para problema mono-objetivo com restrições- *Evolpit*

Inicialmente foram realizados testes considerando um modelo de recursos de uma seção com o número máximo de blocos igual a 4. Primeiramente foi utilizado o fator *R* de penalização na função de mérito, conforme descrito no item 2.4.7.

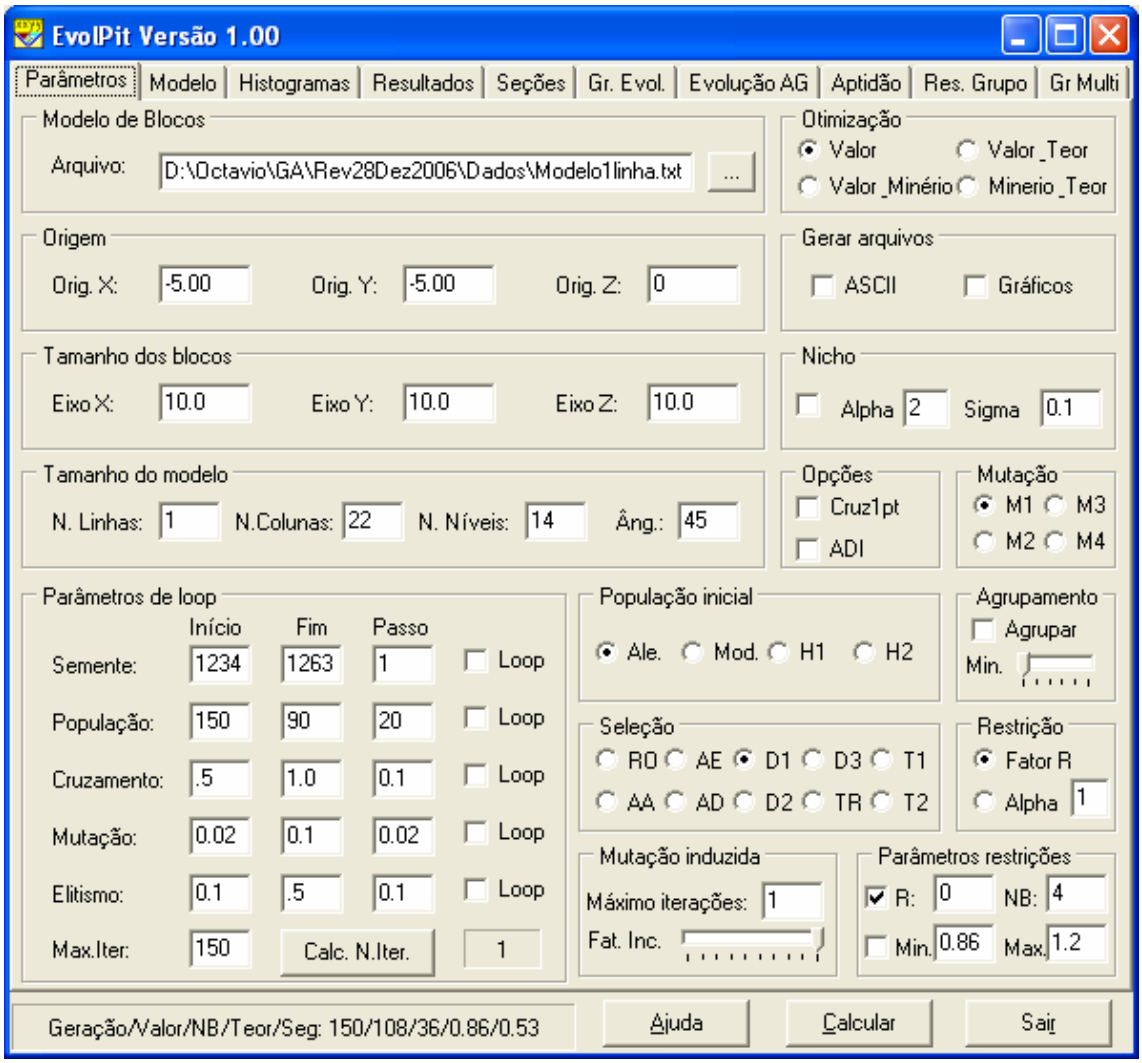


Figura 5.3.1 – Parâmetros *Evolpit* com restrições de massa (fator R).

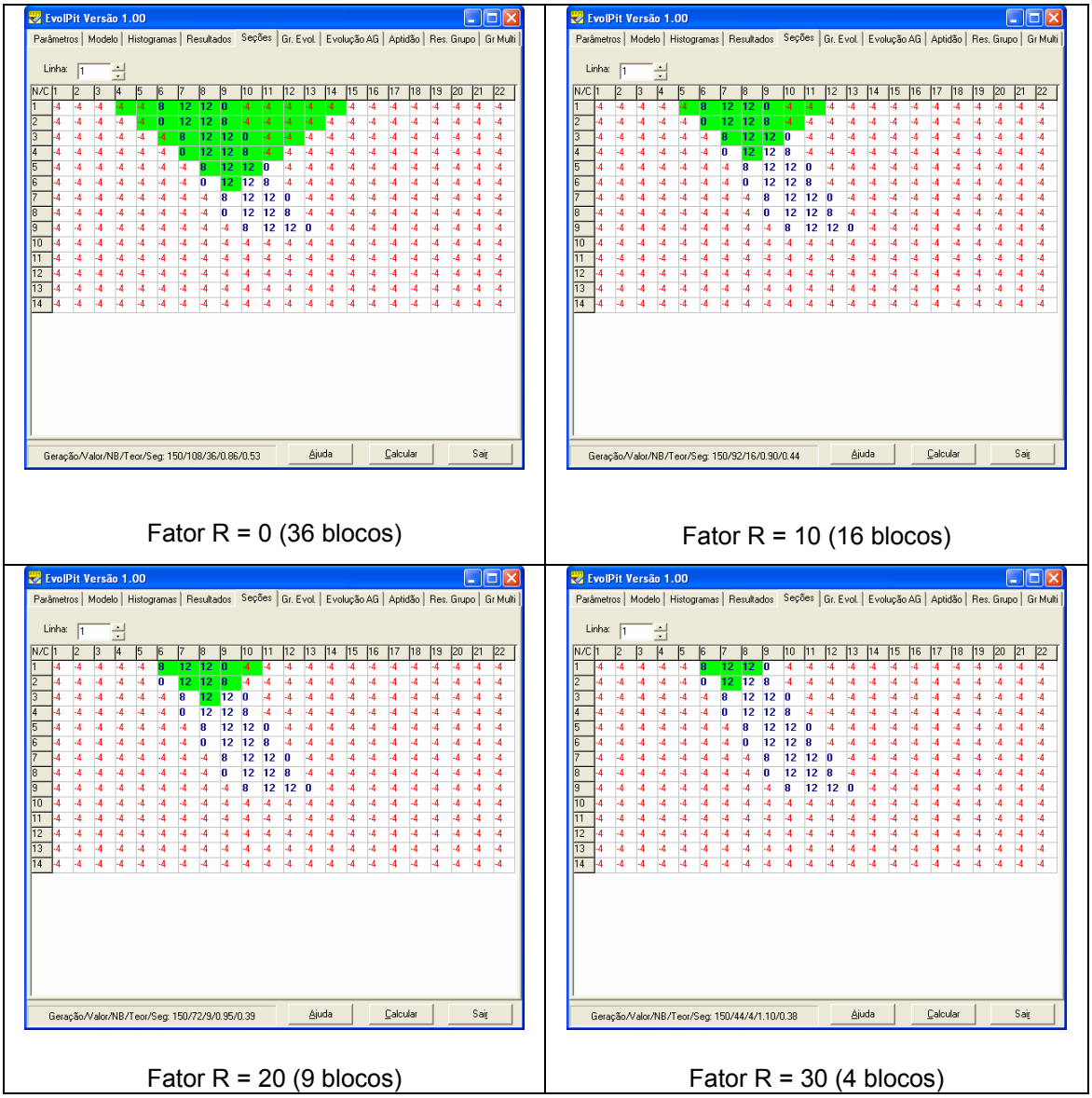
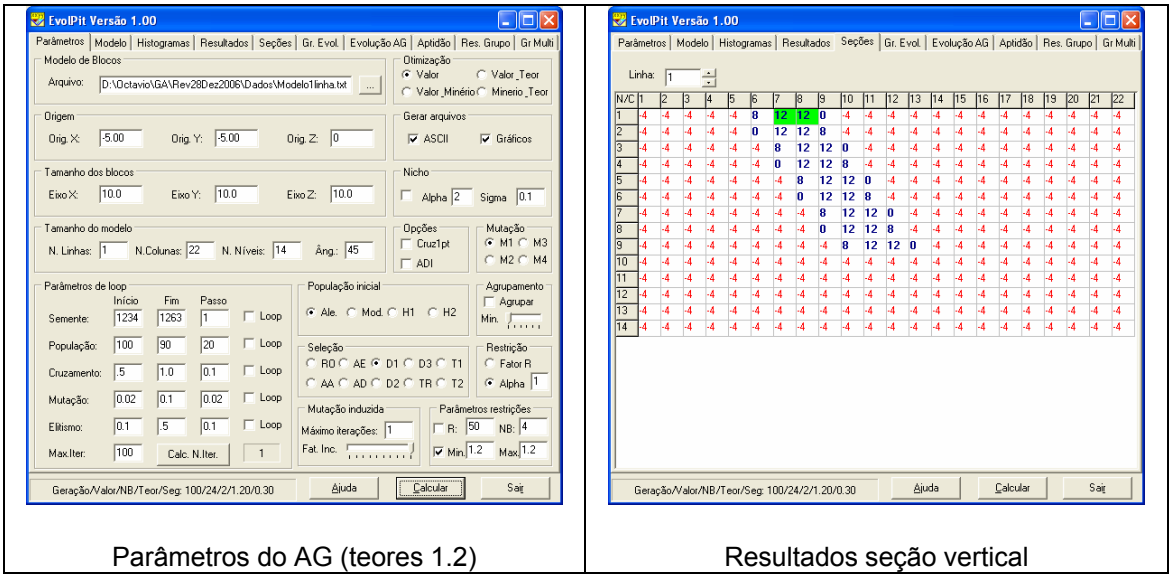


Figura 5.3.2 – Resultados com restrições (fator R).

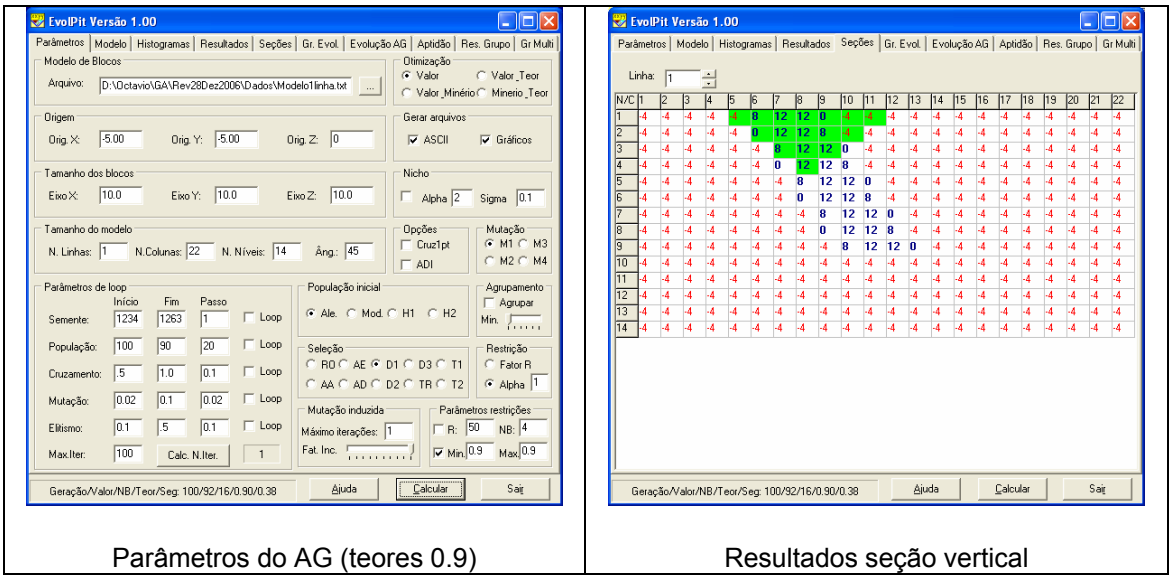
Como se esperava, à medida que o fator de penalização *R* aumentou, o algoritmo tendeu a convergir para a solução correta que, no caso, era de quatro blocos positivos. Tal modelo presta-se à representação da necessidade no seqüenciamento de lavra com relação à quantidade de massa. No caso, está sendo considerada a quantidade de blocos.

Com o objetivo de simular a procura por uma qualidade específica de teor médio, foram estabelecidos como limites máximos e mínimos os teores de 1.2 e 0.9 unidades. Abaixo, os parâmetros utilizados bem como os resultados encontrados:



Parâmetros do AG (teores 1.2) | Resultados seção vertical

Figura 5.3.3 – Resultados *Evolpit* com restrições de teor (1.2).



Parâmetros do AG (teores 0.9) | Resultados seção vertical

Figura 5.3.4 – Resultados *Evolpit* com restrições de teor (0.9).

Conforme se observa, para ambos os limites estabelecidos de teores (1,2 e 0,9), o algoritmo conseguiu convergir para as soluções corretas.

5.3.1 Conclusões dos resultados para problema mono-objetivo com restrições-Evolpit

Por meio das opções disponíveis na interface do programa *Evolpit* pode-se definir o número máximo de blocos que podem estar contidos dentro da cava final visando simular restrições de massa em um seqüenciamento de lavra; além da definição de um intervalo de teores entre um valor mínimo e um valor máximo, cujo objetivo é controlar a qualidade específica de um determinado bem mineral ou contaminante.

Uma vez definido(s) o(s) tipo(s) de restrição(ões) e como será tratada a restrição na função objetivo, ou seja, o valor do parâmetro R (fator de penalização), foi possível gerar cenários de seqüenciamento de lavra controlando-se as massas e teores para uma seção.

5.4 Resultados do algoritmo genético multiobjetivo NSGA - *VBA*

Inicialmente foi implementado um algoritmo para as funções Schafferf2 e Schafferf3 disponíveis na literatura.

Validado o algoritmo, procedeu-se à implementação do programa de maximização do benefício para os problemas propostos no algoritmo genético simples (AGS) com elitismo.

5.4.1 Testes e resultados encontrados das funções de teste

Foram realizados vários testes com diferentes conjuntos de parâmetros conforme abaixo:

a) Tamanho da população: 40, 100, 200 e 250 com uma

população externa sempre com o valor em dobro (80, 200, 400 e 500).

b) Número de gerações: 12, 120, 200 e 2000

c) Probabilidade de cruzamento: 0.9

d) Probabilidade de mutação: 0.01 e 0.08

e) Percentagem de elitismo: 30% e 90%

O melhor resultado é apresentado a seguir:

Parâmetros utilizados: (população maior, número de gerações e elitismo maior)

Tamanho da população:	100
Tamanho da população externa (elitismo):	200
Número máximo de gerações:	120
Probabilidade de cruzamento [0,1]:	0.9
Probabilidade de mutação [0,1]:	0.01
Semente:	12345
Percentagem de elitismo [0,1]:	0.9
Limite mínimo Schafferf2:	-12
Limite máximo Schafferf2:	12
Limite mínimo Schafferf3:	-2
Limite máximo Schafferf3:	10

Resultados:

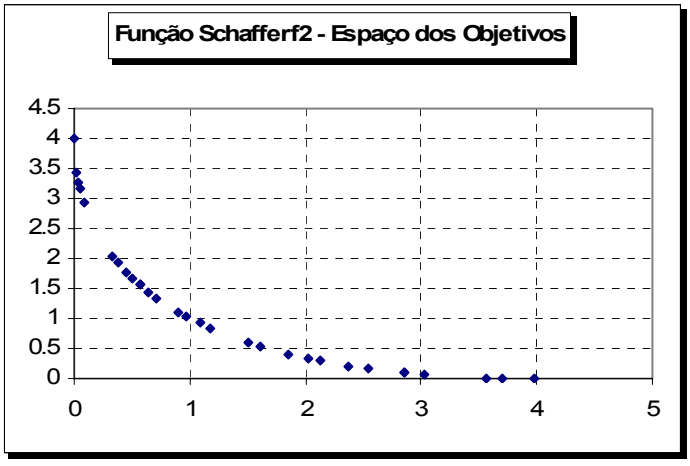


Figura 5.4.1.1 – Gráfico dos resultados função *Schafferf2.*

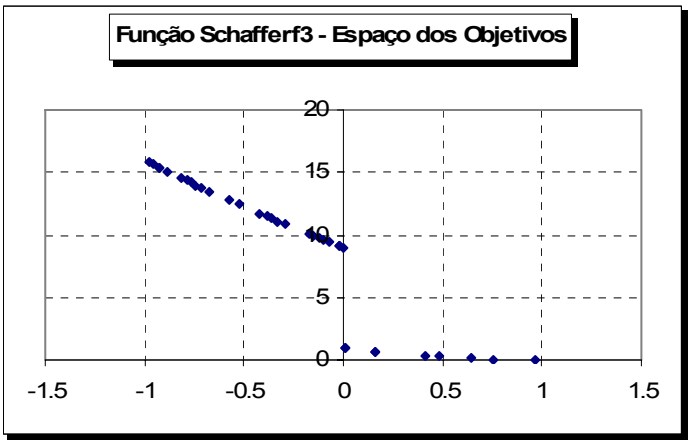


Figura 5.4.1.2 – Gráfico dos resultados função *Schafferf3.*

Nesse teste gerou-se um conjunto de soluções próximas ao conjunto Pareto-Ótimo para as duas funções analisadas de maneira eficiente.

5.4.1.1 Conclusões dos resultados encontrados das funções de teste

O algoritmo genético multiobjetivo com elitismo, baseado no método NSGA (Non-dominated Sorting Genetic Algorithm), originalmente desenvolvido por Srinivas & Deb (1993) e adaptado por ex-alunos da UFMG, ora implementado, mostrou-se eficaz na resolução das duas funções de teste analisadas. Porém exige do projetista uma análise dos resultados encontrados.

Percebe-se, ainda, que o tamanho da população influi fortemente nos resultados, ou seja, com populações menores, o risco de o algoritmo gerar soluções não pertencentes ao conjunto Pareto-Ótimo é maior.

Com relação à identificação dos melhores parâmetros utilizados para os limites adotados, o teste ora apresentado foi eficiente para os dois primeiros problemas. Para a definição desses parâmetros foram necessários vários testes. Portanto, para utilização desse algoritmo envolvendo outros problemas, é interessante que se faça um conjunto razoável de testes com diferentes parâmetros a fim de se encontrar o conjunto de soluções não dominadas.

5.4.2 Resultados do algoritmo NSGA problema multiobjetivo - *VBA*

O objetivo desses testes é avaliar a aplicação do algoritmo genético multiobjetivo NSGA com elitismo que, a partir de modelos de blocos tecnológicos conhecidos, que representem uma jazida, seja capaz de gerar um conjunto de soluções ótimas, tendo em vista os objetivos de maximização de reservas e maximização de valor da cava final.

Foram executados vários testes com diferentes combinações de parâmetros. A seguir, são apresentados os melhores resultados onde são exibidos os gráficos com os pontos encontrados para cada conjunto de parâmetros referente ao modelo de recurso de uma seção vertical.

Tamanho da população [1:250]:	250
Tamanho da população externa (elitismo):	500
Número máximo de gerações:	250
(1) Mono-objetivo; (2) = Multiobjetivo:	2
Probabilidade de cruzamento [0,1]:	0.9
Probabilidade de mutação [0,1]:	0.02
Semente:	1234
Percentagem de elitismo [0,1]:	0.9
Fator de partilha: (0) automático;	0
Ponderador Valor [0,1]:	1
Ponderador Massa [0,1]:	0
Número de seções [1,22]:	1

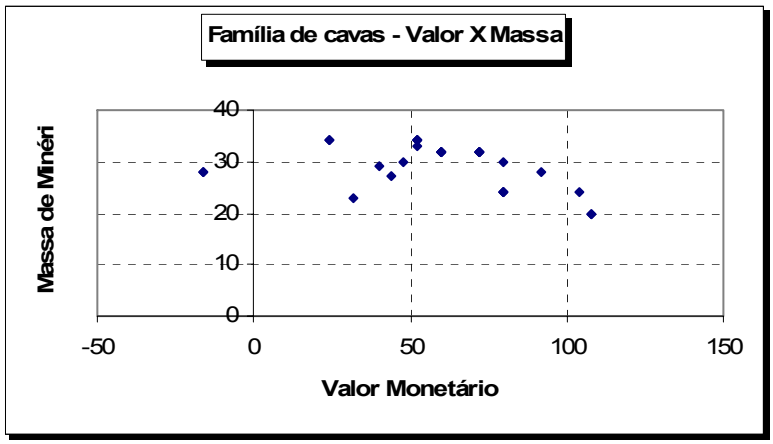


Figura 5.4.2.1 – Gráfico espaço dos objetivos NSGA multi (250 Gerações - 1 seção) – *VBA*.

Conforme pode ser observado, o algoritmo conseguiu gerar um conjunto de cavas ótimas que maximizam dois objetivos.

A seguir, são apresentados os resultados dos testes para duas seções onde o tamanho da população utilizado foi de 250 indivíduos:

Tamanho da população [1:250]:	250
Tamanho da população externa:	500
Número máximo de gerações:	250
(1) Mono-objetivo; (2) = Multiobjetivo:	2
Probabilidade de cruzamento [0,1]:	0.9
Probabilidade de mutação [0,1]:	0.02
Semente:	1234
Percentagem de elitismo [0,1]:	0.9
Fator de partilha: (0) automático:	0
Ponderador Valor [0,1]:	1.0
Ponderador Massa [0,1]:	0
Número de seções [1,22]:	2

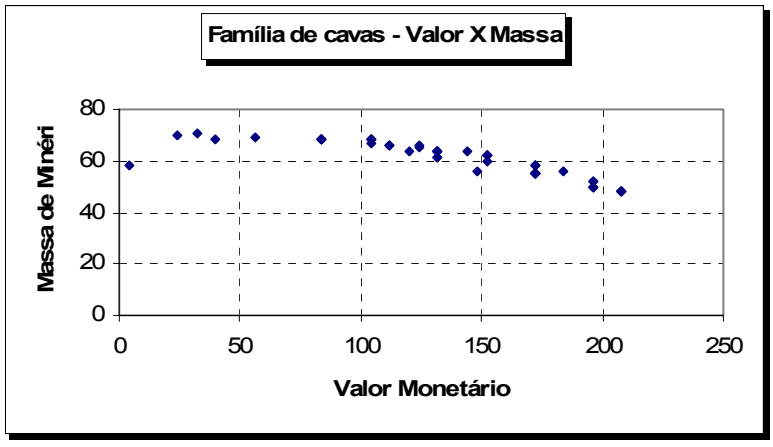


Figura 5.4.2.2 – Gráfico espaço dos objetivos NSGA (250 Gerações - 2 seções) – *VBA.*

Nesse teste, o algoritmo conseguiu gerar os dois pontos ótimos dos dois objetivos formulados.

Parâmetros utilizados: 3 Seções

Tamanho da população [1:250]:	250
Tamanho da população externa:	500
Número máximo de gerações:	250
(1) Mono-objetivo; (2) = Multiobjetivo:	2
Probabilidade de cruzamento [0,1]:	0.9
Probabilidade de mutação [0,1]:	0.02
Semente:	1234
Percentagem de elitismo [0,1]:	0.9
Fator de partilha: (0) automático:	0
Ponderador Valor [0,1]:	1
Ponderador Massa [0,1]:	0
Número de seções [1,22]:	3

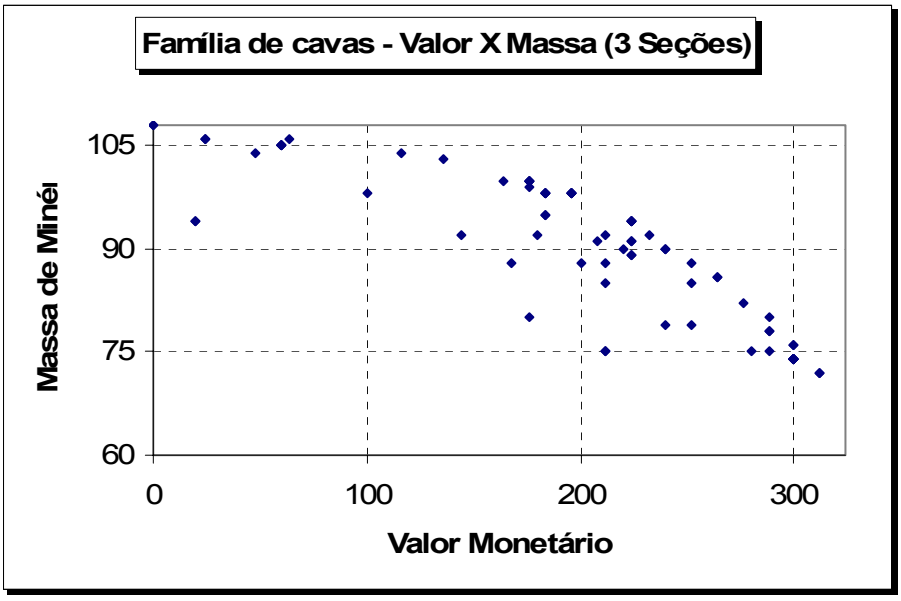


Figura 5.4.2.3 – Gráfico espaço dos objetivos NSGA (250 Gerações - 3 seções) – *VBA*.

Como pode ser observado, o algoritmo conseguiu encontrar a cava que maximiza o valor monetário como também o valor máximo dos recursos.

Parâmetros utilizados: 4 Seções

Tamanho da população [1:250]:	250

Tamanho da população externa:	500
Número máximo de gerações:	250
(1) Mono-objetivo; (2) = Multiobjetivo:	2
Probabilidade de cruzamento [0,1]:	0.9
Probabilidade de mutação [0,1]:	0.02
Semente:	1234
Percentagem de elitismo [0,1]:	0.9
Fator de partilha: (0) automático:	0
Ponderador Valor [0,1]:	1
Ponderador Massa [0,1]:	0
Número de seções [1,22]:	4

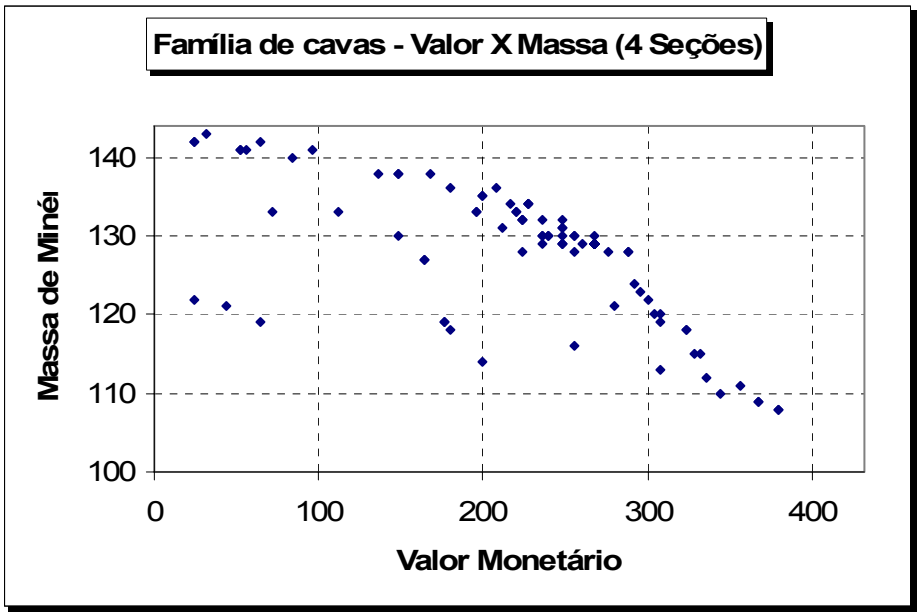


Figura 5.4.2.4 – Gráfico espaço dos objetivos NSGA (250 Gerações - 4 seções) *VBA.*

Como pode ser observado, o algoritmo não conseguiu encontrar a cava que maximiza o valor monetário. O valor máximo encontrado foi de 380 unidades monetárias que corresponde a 90% da solução ótima.

Parâmetros utilizados: 6 Seções

Tamanho da população [1:250]:	250
Tamanho da população externa:	500

Número máximo de gerações:	250
(1) Mono-objetivo; (2) = Multiobjetivo:	2
Probabilidade de cruzamento [0,1]:	0.9
Probabilidade de mutação [0,1]:	0.02
Semente:	1234
Percentagem de elitismo [0,1]:	0.9
Fator de partilha: (0) automático:	0
Ponderador Valor [0,1]:	1
Ponderador Massa [0,1]:	0
Número de seções [1,22]:	6

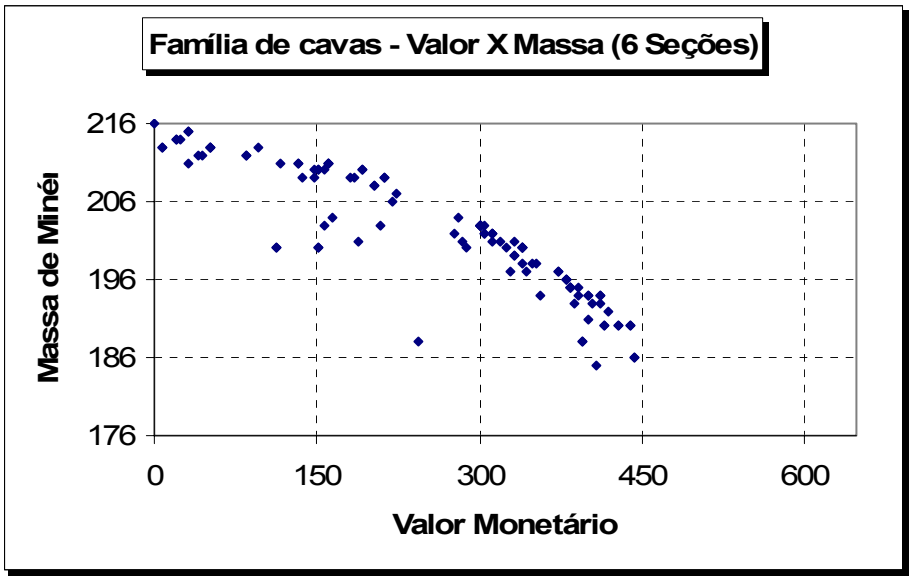


Figura 5.4.2.5 – Gráfico espaço dos objetivos NSGA (250 Gerações - 6 seções) *VBA.*

Como se observa, o algoritmo não conseguiu encontrar a cava que maximiza o valor monetário. O valor máximo encontrado foi de 444 unidades monetárias que corresponde a 69% da solução ótima. Já para a maximização dos recursos, a solução ótima foi encontrada no indivíduo 79 que possui 216 unidades de massa.

Parâmetros utilizados: 22 Seções

Tamanho da população [1:250]:	250
Tamanho da população externa:	500
Número máximo de gerações:	250
(1) Mono-objetivo; (2) = Multiobjetivo:	2

Probabilidade de cruzamento [0,1]:	0.9
Probabilidade de mutação [0,1]:	0.02
Semente:	1234
Percentagem de elitismo [0,1]:	0.9
Fator de partilha: (0) automático:	0
Ponderador Valor [0,1]:	1
Ponderador Massa [0,1]:	0
Número de seções [1,22]:	22

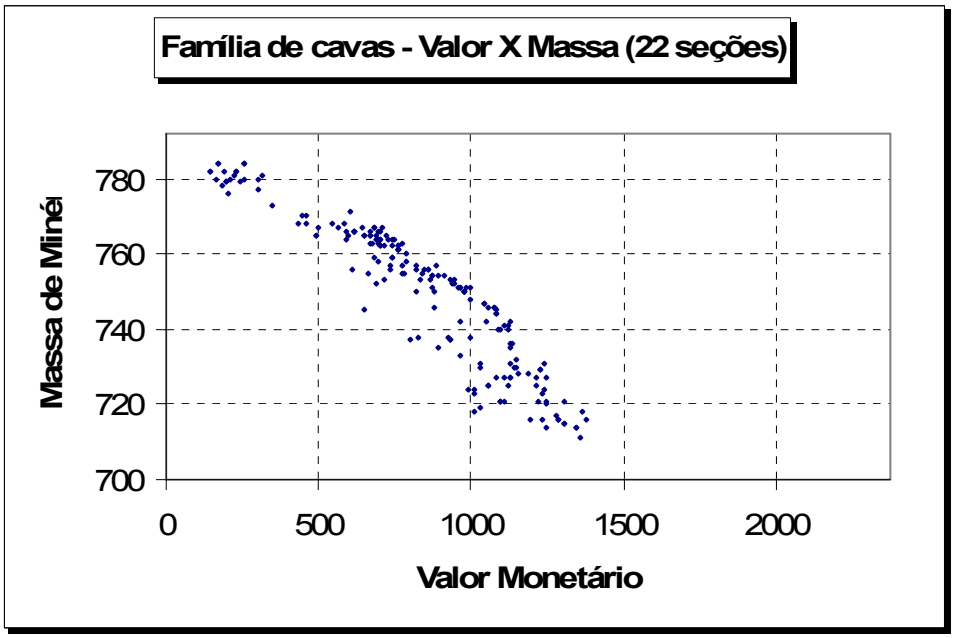


Figura 5.4.2.6 –Gráfico espaço dos objetivos NSGA (250 Gerações - 22 seções) *VBA.*

O algoritmo não conseguiu encontrar a cava que maximiza o valor monetário. O valor máximo encontrado foi de 376 unidades monetárias que correspondem a 58% da solução ótima Já para a maximização dos recursos obtiveram-se 784 unidades de massa, que correspondem a 99% da solução ótima.

5.4.2.1 Conclusões do algoritmo NSGA problema multiobjetivo *VBA*

O algoritmo genético multiobjetivo com elitismo, baseado no método NSGA (Non-dominated Sorting Genetic Algorithm), originalmente desenvolvido por Srinivas &

Deb em 1993 e adaptado por ex-alunos da UFMG, ora implementado, mostrou ser capaz de gerar famílias de cavas ótimas para modelos com até três seções de blocos.

A seguir, o gráfico com a família ótima de cavas obtidas para duas seções.

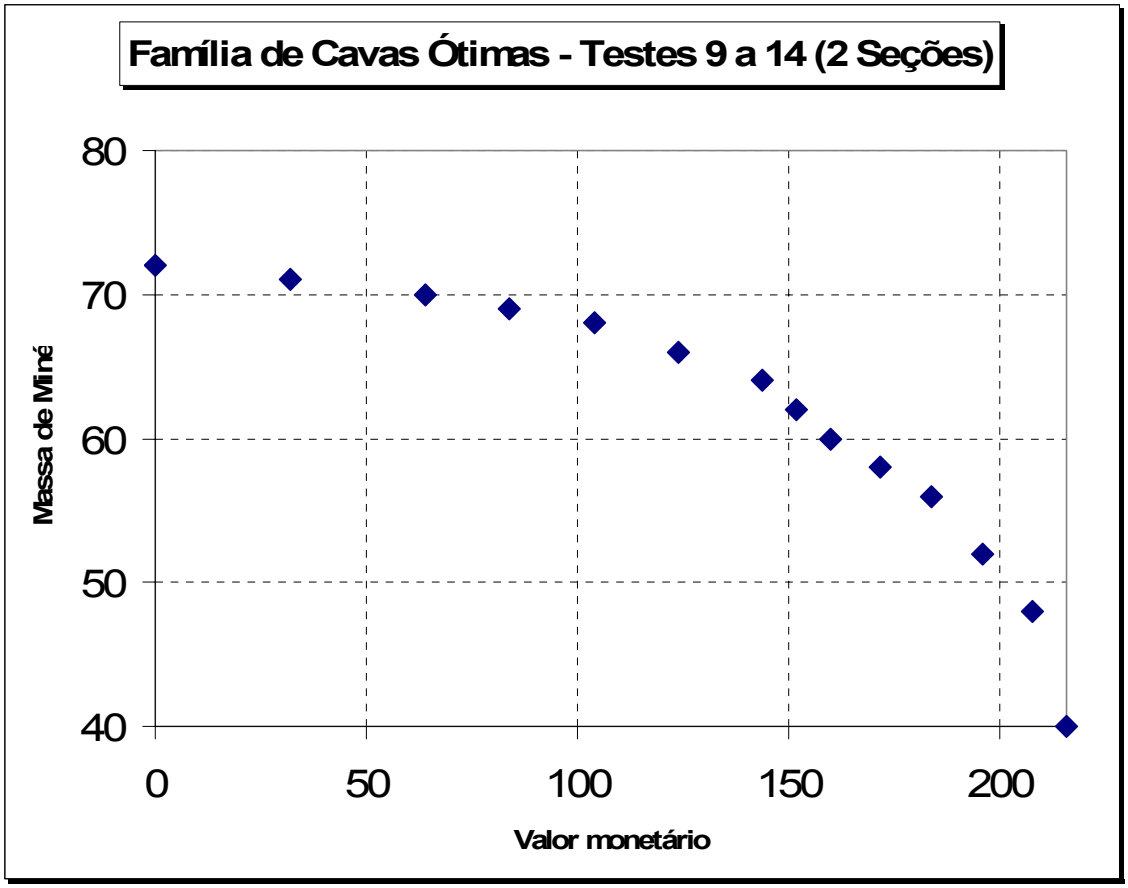


Figura 5.4.2.1.1 – Gráfico espaço dos objetivos NSGA multi (250 Gerações – 2 seções) *VBA.*

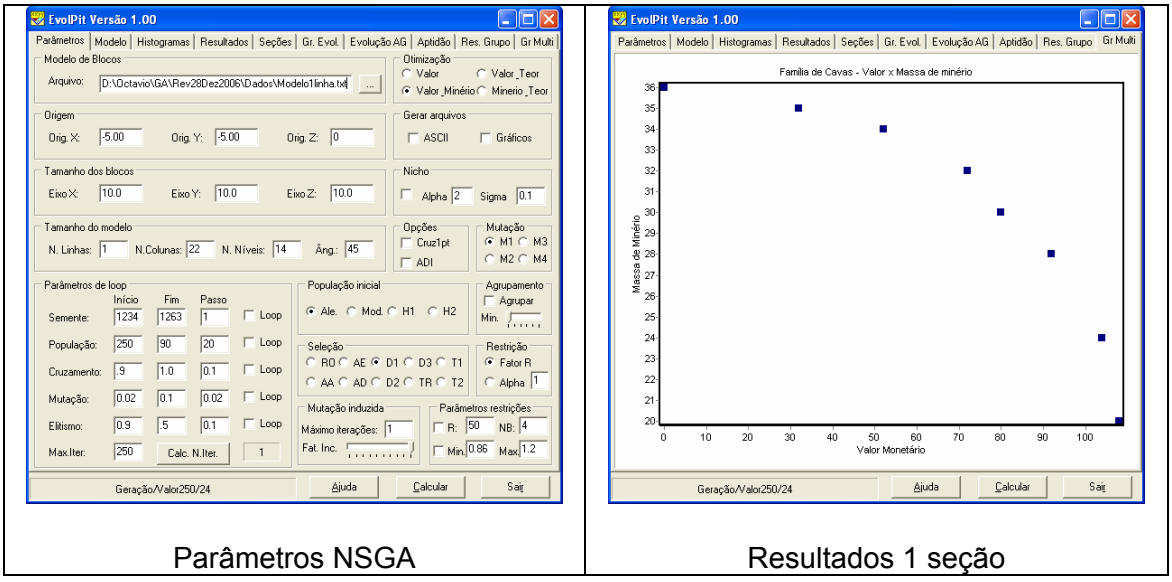
Como se observa no gráfico para duas seções, o algoritmo conseguiu gerar, de maneira eficiente, um conjunto de famílias de cavas que maximiza o lucro não descontado e as reservas.

Para um número maior de seções e considerando-se o objetivo de maximização de recursos, o algoritmo conseguiu gerar um conjunto razoável de pontos. Porém, para o objetivo de maximização do lucro, percebe-se que o algoritmo NSGA não conseguiu gerar cavas próximas das soluções ótimas.

5.4.3 Resultados do algoritmo NSGA problema multiobjetivo Evolpit - *Delphi*

Conforme descrito no item 4.4.2.3, além dos objetivos de maximização de recursos e valor monetário, foram implementados os pares de objetivos (Valor Monetário X Teor) e (Recursos x Teor).

A seguir, figura contendo os parâmetros e os resultados encontrados no programa *Evolpit* para o problema multiobjetivo de maximização dos recursos e o valor monetário considerando apenas uma seção vertical.

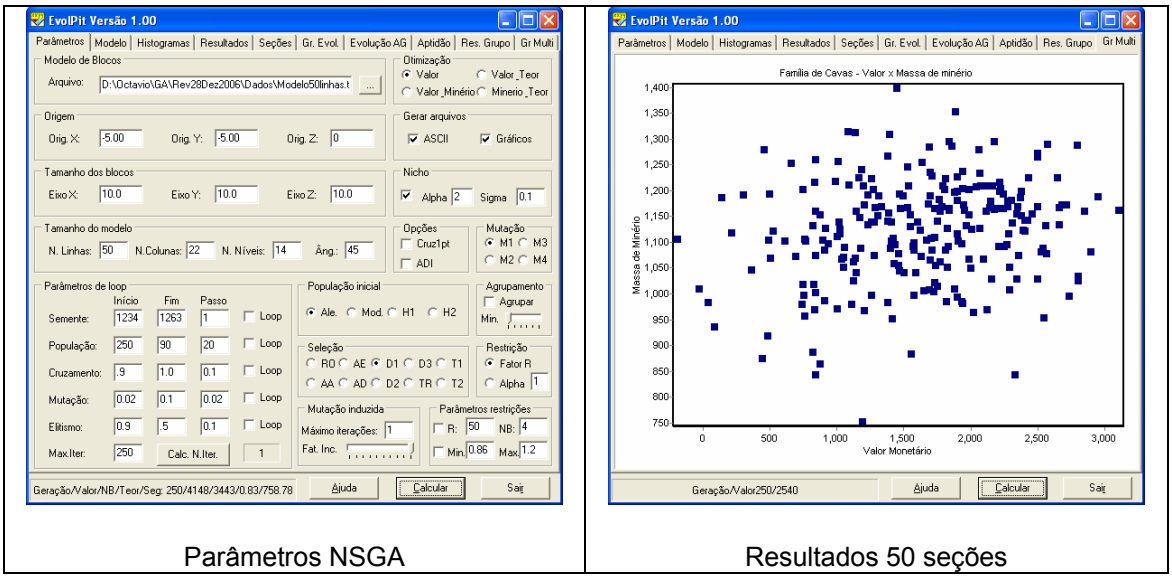


Parâmetros NSGA | Resultados 1 seção

Figura 5.4.3.1 – Parâmetros e resultados NSGA Recursos/Valor 1 seção (*Evolpit*).

Observa-se que o algoritmo foi capaz de gerar um conjunto de cavas ótimas tendo em vista os dois objetivos formulados para o modelo de recursos contendo uma seção vertical.

Foi simulado ainda um modelo de recursos contendo 50 seções verticais. Ressalta-se que, no *VBA,* tal simulação não seria possível, tendo em vista o limite de 64k de memória.

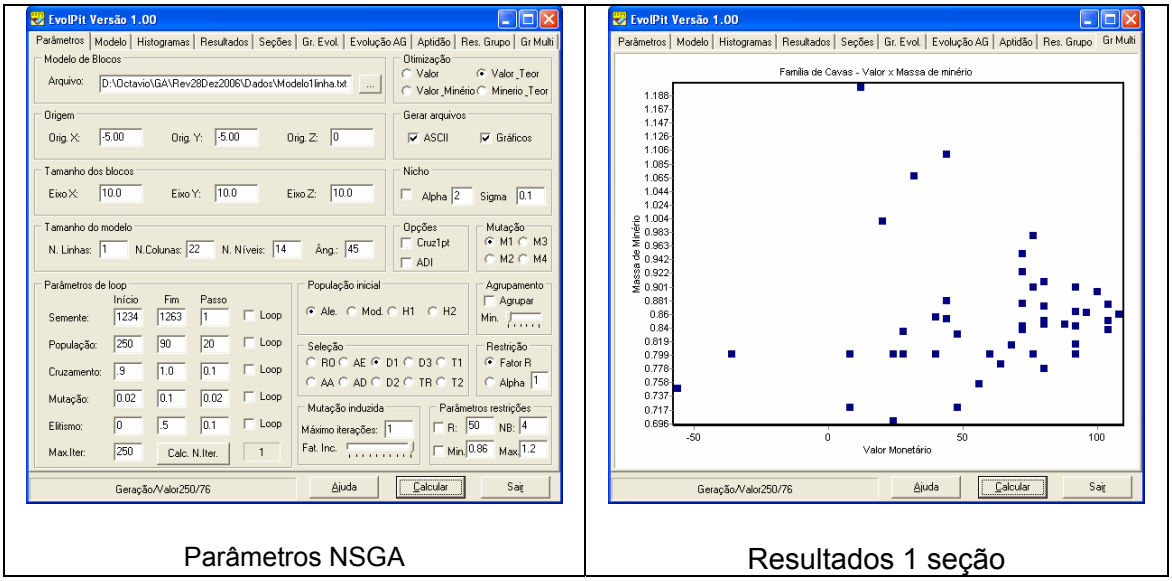


Parâmetros NSGA | Resultados 50 seções

Figura 5.4.3.2 – Parâmetros e resultados NSGA Recursos/Valor 50 seções (*Evolpit*).

Conforme se observa, o algoritmo atingiu 3108 unidades monetárias e 1398 unidades de massa que correspondem respectivamente a 58% e 78% das soluções ótimas dos dois objetivos respectivamente.

A seguir, parâmetros e resultados encontrados para a otimização multiobjetivo de valor monetário e massa para o modelo de recursos de 1 seção.

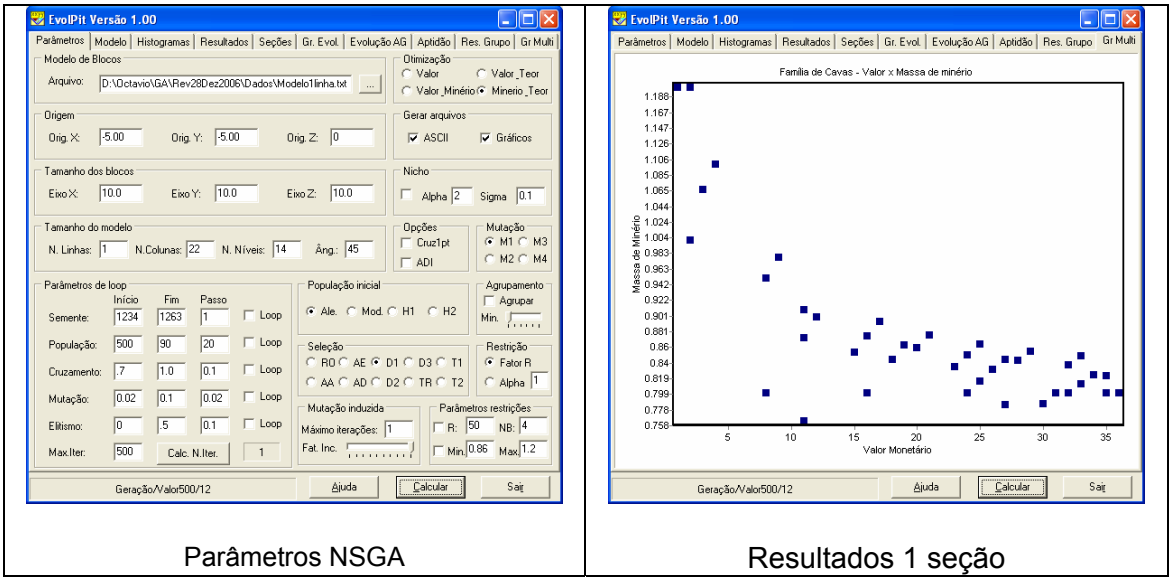


Parâmetros NSGA | Resultados 1 seção

Figura 5.4.3.3 – Parâmetros e resultados NSGA Valor/Teor 1 seção (*Evolpit*).

Percebe-se que o algoritmo conseguiu convergir para as duas soluções ótimas, porém gerou relativamente poucas cavas ótimas quando comparado à otimização Recursos X Valor.

Foi simulada, ainda, a otimização considerando-se os objetivos de recursos e teor médio para o modelo de uma seção, conforme abaixo:



Parâmetros NSGA | Resultados 1 seção

Figura 5.4.3.4 – Parâmetros e resultados NSGA Recursos/Teor 1 seção (*Evolpit*).

O algoritmo conseguiu gerar um conjunto de cavas ótimas tendo em vista os dois objetivos de maximização dos recursos e teor médio para o modelo de 1 seção.

5.4.3.1 Conclusões do algoritmo NSGA problema multiobjetivo Evolpit – *Delphi*

Tendo em vista as opções de otimização para os problemas multiobjetivos propostos: Recursos x Valor, Recursos x Teor e Valor x Teor, foi possível gerar várias famílias de cavas ótimas utilizando-se o programa Evolpit.

São necessários testes adicionais, principalmente para modelos de recursos maiores com o objetivo de validar sua aplicabilidade do mesmo em problemas reais.

6 Conclusões e trabalhos futuros

Todos os algoritmos implementados, para os problemas analisados, conseguiram convergir para as soluções teóricas conhecidas para um número reduzido de blocos. O tamanho da população e o número de gerações influíram fortemente nos resultados, ou seja, com populações menores, o risco de o algoritmo convergir para máximos locais é maior, bem como a geração de pontos dominados na fronteira do Pareto-Ótimo.

Para o algoritmo genético simples com elitismo e o NSGA desenvolvidos, foi possível identificar uma combinação de parâmetros (probabilidade de cruzamento, mutação e elitismo), para os problema propostos. Ressalta-se que tais valores são válidos somente para os problemas analisados, tendo em vista ainda a quantidade de blocos de minério no modelo de blocos utilizado.

O algoritmo genético multiobjetivo com elitismo, baseado no método NSGA (Non-dominated Sorting Genetic Algorithm), exige que se façam vários testes e uma análise *a posteriori* dos resultados encontrados.

E interessante que se façam testes utilizando-se diferentes modelos de recursos e reservas reais e que se comparem os resultados obtidos com os encontrados em outros sistemas disponíveis no mercado. Tais testes poderão realizar-se, tanto para determinação dos limites de cava final, quanto para seqüenciamento de lavra considerando-se metas de massas e teores.

Com pequenas alterações no código do programa de computador *Evolpit* é possível utilizá-lo para validação de modelos já simulados, pois existe a opção de se ler um modelo com os limites de cava final previamente definido na geração da população inicial.

Sugere-se investigar, implementar e testar outros algoritmos de agrupamento, seleção, mutação e adaptação dinâmica, no intuito de melhorar os resultados do programa, tendo em vista modelos de recursos e reservas maiores.

7 Referências bibliográficas

BAKER, J.E. *Adaptive Selection Methods for Genetic Algorithms*. Proceedings of the First International Congress on Genetic Algorithms, pp. 101-111, 1985.

BARNES, R.J. *Optimizing the ultimate pit*. Colorado School of Mines, MSc. Thesis, 1982. 120p, 1982.

BONGARÇON, D.F. e MARÉCHAL, A. *A New Method for Open-Pit Design Parametrization of the Final Pit-Contour*, 14th APCOM, Pennsylvania, p. 573-583, 1976.

CARMO, F. A. R. *Metodologias para o Planejamento de Cavas Finais de Minas a Céu Aberto Otimizadas*. Dissertação de Mestrado - UFOP. Ouro Preto, 2001.

COELHO, C. A. C. e TOSCANO, G. *A micro-genetic algorithm for multi-objective optimization*. Technical Report Lania-RI-2000-06, Laboratoria Nacional de Informatica Avanzada, Xalapa, Veracruz, Mexico, 2000.

CORNE, D., KNOWLES, J. e OATES, M. *The Pareto envelope-based selection algorithm for multiobjective optimization*. Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature VI (PPSN-VI), pp. 839-848, 2000.

DAGDALEN, K. *Open Pit Optimization – Strategies for Improving Economics of Mining Projects Through Mine Planning*. 17th International Mining Congress of Turkey, Ankara. The Cahmber of Mining Engineers of Turkey. pp 117-122, 2001.

DAVID, M., P. DOWD e KOROBOV S. *Forecasting departure from planning in open pit design and grade control*. APCOM, Golden, CO. 1974.

DAVIS, L. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold. New York, 1991.

DEB, K. *Genetic Algorithms in multimodal function optimization*. Master Thesis, TGGA report n. 89002. University of Alabama, 1998.

DEB, K. *Evolutionary Algorithms for Multi-Criterion Optimization in Engineering Design*. Evolutionary Algorithms in Engineering and Computer Science, cap. 8, pp.135-161. John Wiley & Sons, Ltd. Chichester, UK, 1999.

DEB, K., AGRAWAL, S., PRATAP, A. e MEYARIVAN, T. *A fast and elitist multi-objective genetic algorithm: NSGA-II*. Technical Report 200001, Indian Institute of Technology, Kanpur: Kanpur Genetic Algorithm Laboratory (KanGAL), 2000.

DEB, K. *Multi-Objective Optimization using Evolutionary Algorithms*. 1 ed. England, John Wiley & Sons, 2001. 515p.

DIAS, A.H.F. *Algoritmos Genéticos Aplicados a Problemas com Múltiplos Objetivos*. UFMG 2000, 136p. (Dissertação, Mestrado em Engenharia Elétrica).

FONSECA, C. M. e FLEMING, P. J. *Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization*. Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 416-423. 1993.

FORD, L. R. e FULKERSON D. R. *Maximum flow through a network*. Canadian Journal of Mathematics, vol. 8: pp. 399-404, 1956.

GOLDBARG, M.C. e LUNA, H.P.L. *Otimização combinatória e programação linear*. 2 ed. Rio de Janeiro: Elsevier Editora Ltda, 2005. 518p.

GOLDBERG, D.E. e RICHARDSON, J. *Genetic algorithm with sharing for multimodal function*. Proceedings of the Second International Conference on Genetic Algorithms, pp 41-49. Lawrence Erlbaum, 1987.

GOLDBERG, D.E. *Genetic Algorithm in Search, Optimization and Machine Learning*. Reading, Addison-Wesley. USA, 1989.

GOLDBERG, D.E. e DEB, K. *A comparison of selection schemes used in genetic algorithms*. Foundations of Genetic Algorithms, pp. 69-93. Morgan Kaufmann, San mateo, California, 1991.

HAJELA, P. e LIN, C.Y. *Genetic Serach Strategies in multi-criterion optimal design*. Structural Optimization, 4(2), pp. 99-107. 1992.

HALATCHEV, R. *The Time Aspect of the Optimum Long-Term Open Pit Production Sequencing*. 30th. Application of Computers and Operations Research in the Mineral Industry, Littletown, SME. 2002.

HOLLAND, J.H. *Adaptation in natural and Artificial Systems*. Ann Arbor, University of Michigan Press. USA, 1975.

HORN, J., NAFPLOITIS, N. e GOLDBERG, D. *A niched Pareto genetic algorithm for multi-bjective optimization*. Proceedings of the First IEEE Conference on Evolutionary Computation, pp. 82-97, 1994.

HORN, J. *Muticriterion decision making*. Handbook of Evolutionary Computation, volume I, pp. F1.9:1-F1.9:15. IOP Publishing Ltd. And Oxford University Press, 1997.

HUSTRULID, W. e KUCHTA, M. *Open Pit Mine Planning & Design*. 1. Rotterdam: A.A. Balkema, 1995, 636p.

HUTTAGOSOL, P. e CAMERON R. E. *A computer design of ultimate pit limit by using transportation algorithm*. 23rd APCOM, Tucson. pp. 443-460, 1992.

JOINES, J.A. e HOUCK, C.R. *On the use of nonstationary penalty functions to solve nonlinear constrained optimization problems with GAs*. Proceedings of the International Conference on Evolucionary Computation, pp. 579-584, 1994.

JOHNSON, T.B. e SHARP, W.R. *A three dimensional dynamic programming method for optimal ultimate open-pit design*. U.S. Bureau of Mines Report of Investigations. 1971.

KIM, Y.G. *Open Pit Limit Analysis*. Computer Methods for the 80's in the Mineral Industry, SME-AIME, pp. 297-303, 1978.

KITA, H., YABUMOTO, Y., MORI, N. e NISHIKAWA, Y. *Multi-objective optimization by means of thermodynamical genetic algorithm*. Proceedings of Parallel Problem Solving from Nature IV (PPSN-IV), pp. 504-512, 1996.

KNOWLES, J. D. e CORNE, D. W. *Approximating the non-dominated front using the Pareto archived evolution strategy*. Evolutionary Computation Journal 8(2), 149-172, 2000.

LANE, K. *The Economic Definition of Ore*. London: Mining Journal Books, 1988.

LAUMANNS, M., RUDOLPH, G. e SCHWEFEL, H. P. *A spatial predator-prey approach to multi-objeticve optimization: A preliminary study*. Proceedings of the Parallel Problem Solving from Nature V (PPSN-V), pp. 241-249, 1998.

LEMIEUX, M. *Moving cone optimizing algorithm*. Computer methods for the 80s in the mining industry, SME. pp. 329-345, 1979.

LERCHS, H. e GROSSMANN I. F. *Optimum design of open pit mines*. CIM Bulletin, vol. 58 (January), pp. 47-54, 965.

MATHERON, G. *Le paramétrage des contours optimaux*. Technical notes 401 and 403, Centre de Géostatistique de l'Ecole des Mines de Paris, Fontainebleau, France, 19p. et 5 4p., 1975.

MICHALEWICZ, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag. New York, 1992.

NORONHA, R. A. *Tomada de Decisão em Projeto de Mineração: Quantificação de Riscos e Incertezas*. Dissertação de Mestrado - UFMG. Belo Horizonte, 2001.

OSYCZKA, A. *Multicriteria optimization for engineering design*. In John S. Gero, editor, Design Optimization, pp. 193-227. Academic Press, 1985.

PANA, M. T. e CARLSON T. R.. *Description of a Computer Technique Used in Mine Planning of the Utah Mine of Kennnecott Copper Corp.* 6th APCOM. 1966.

PARETO, V. *Cours D'Economic Politique*. Volume I e II. F. Rouge, Lausanne, 1896.

PERONI, R. L. *Análise de sensibilidade do sequenciamento de lavra em função da incerteza do modelo geológico*. Tese de Doutorado – UFRS. Porto Alegre, 2002.

RUDOLPH, G. *Evolutionary search under partially ordered fitness sets*. Proceedings ot the International Symposium on Information Science Innovations in Engineering of Natural and Artificial Intelligent Systems (ISI), pp. 818-822, 2001.

SAINSBURY, G.M. *Computer-based design of open cut mines*. Australasian Institute of Mining and Metallurgy Conference. Pp 49-57. 1970.

SAMANTA, B. e BHATTACHERJEE, A. *A genetic algorithms approach for grade control planning*. APCOM, 2005.

SAYDAM, S. e YALCIN, E. *Reserve and Ultimate Pit Limit Design Analisys of Caldagi Nickel Deposit, Turkey*. 30th Application of Computers and Operations Research in the Mineral Industry, Littletown, SME. Pp 121-131, 2002.

SCHAFFER, J. D., *Multiple objective optimization with vector evaluated genetic algorithms*. Proceedings of the First International Conference on Genetic Algorithms, pp93-100, 1985.

SRINIVAS, N. e DEB, K. *Multiobjective optimization using nondominated sorting in genetic algortihms*. Technical report, Departament of Mechanical Engineering, Indian Institute of Technology, Kanput. Índia, 1993.

SRINIVAS, G. e PATNAIK, L.M. *Genetic Algorithms: A Survey*. IEEE Computer, v.27, n.6, pp.17-26, 1994.

SYSWERDA, G. *Uniform Crossover in Genetic Algorithms*. Proceedings of the Third International Congress on Genetic Algorithms, pp.2-9, 1989.

SYSWERDA, G. *A Study of Reproduction in Generational and Steady-State Genetic Algorithm*. Foundations of Genetic Algorithms, pp. 94-101. Morgan Kaufmann, 1991.

TAKAHASHI, R. H. C. *Otimização Escalar e Vetorial*. Notas de aula UFMG. Belo Horizonte, 2004.

TANOMARU, J. *Motivação, Fundamentos e Aplicações de Algoritmos Genéticos*. II Congresso de Redes Neurais. Curitiba, 1985.

THOMAS, G.S. *Optimization and Scheduling of Open Pits via Genetic Algorithms and Simulated Annealing*. Proceedings 1st International Symposium on Mine Simulation via the Internet, Paper TG085A. 1996.

TOLEDO, J. P. *Utilização do algoritmo genético aplicado ao planejamento de lavra.* Monografia MBA – USP. São Paulo, 2003.

TOLWINSKI, B. e UNDERWOOD R. *An Algorithm to Estimate the Optimal Evolution of an Open Pit*. 23rd APCOM. pp. 399-409, 1992.

UNDERWOOD, R. e B. TOLWINSKI. *A mathematical programming viewpoint for solving the ultimate pit problem*. European Journal of Operational Research, (107): pp. 96-107, 1998.

VALLET, R. *Optimization mathématique de l'éxploitation d'une mine à ciel ouvert ou le probléme de envelope*, Annales des mines de Belgique, Février, p.113-135, 1976.

VASCONCELOS, J.A. *Notas de aula da disciplina Computação Evolucionária*. Departamento de Engenharia Elétrica/Eletrônica, UFMG, 2004.

VELDHUIZEN, D. V. Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations. Ph. D. Thesis, Dayton, OH: Air Force Institute of Technology AFIT/DS/ENG/99-01, 1999.

YAMATOMI, J., MOGI, A. e YAMAGUCHI, U. *Selective Extraction Dynamic Cone Algorithm for Three Dimensional Open Pit Designs*. XXV APCOM, pp. 267-274, 1995.

WAGENINGEN, A.V.; DUNN,P.G.;MULDOWEY, D.M. *Sequence optimization for long-term mine planning*. APCOM, 2005.

WANG, Q. e SEVIM, H. *Alternative to parameterization in finding a series of maximum-metal pits for production planning*. Mining Engineering. pp. 178-182, 1995.

ZHAO, Y. e KIM, Y. C. *A New Graph Theory Algorithm for Optimal Pit Design*. SME Transactions, vol. 290:, pp. 1832-1838, 1991.

ZHAO, Y. e KIM, Y. C. *A New Optimum Pit Limit Design Algorithm*. 23rd APCOM, Tucson. pp. 432-434, 1992.

ZITZLER, E. e THIELE, L. *An evolutionary algorithm for multiobjective optimization: The strength Pareto approach*. Thechnical Report 43, Zürich, Switzerland: Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), 1998.

8 Anexos

8.1 Anexo I - Listagem do programa algoritmo genético simples com elitismo

Baseado em Goldberg, D.E. (1989)

```
'EvolPit Vesão 1.00 - Algoritmo Genético Simples com Elitismo

Option Explicit

  Public Const PopMax      As Integer = 500  'Tamanho Máximo da
população
  Public Const CompMax     As Integer = 500  'Tamanho Máximo do
cromossomo
  Public Const NV          As Integer = 1    'Numero de variaveis
  Public Linha             As Integer        'Variável de controle dos
resultados na planilha
  Public Comp_Cromossomo   As Integer        'Tamanho do cromossomo
(número de blocos de minério)
  Public Cold(500)         As Boolean        'Cromossomo antigo
  Public Niter             As Integer        'Número de iterações
  Public ProbPop(500)      As Single         'Probabilidade de selecao
do individuo
  Public NCopias(500)      As Integer        'Número de cópias de cada
individuo
  Public ProbPopAc(500)    As Single         'Probabilidade acumulada
de seleção do indivíduo
  Public CopiaCromos(PopMax, CompMax) As Boolean 'Nova população dos
cromossomos
  Public Populacao(500)    As TIndividuo     'Conjunto de individuos
  Public Lamb              As Single         'Parâmetro de ajuste da
função de aptidão
  Public AGS               As TAGS           'Contem os parâmetros do
AGS
  Public GAParam           As TGAParam       'Contem os parâmetros do
AGS em grupo
  Public BlockModel        As TBlockModel    'Parâmetros do modelo de
blocos
  Public BeginTime         As Date           'Variável de controle de
tempo de processamento incio
  Public EndTime           As Date           'Variável de controle de
tempo de processamento fim
  Public NExec             As Integer        'Número de execuções a
serem realizadas

Type TIndividuo                   'Estrutura do indivíduo
  Cromossomo(CompMax) As Boolean 'Vetor relativo ao valor binário das
variáveis do indivíduo (bloco)
  Valor      As Single           'Valor da função do indivíduo
  Aptidao    As Single           'Valor da aptidão do indivíduo
End Type

Type TAGS
  TamPop   As Integer  'Tamanho da população
  GerMax   As Integer  'Numero de gerações
  Precisao As Single   'Precisão
  PCruz    As Single   'Probabilidade de cruzamento
  PMut     As Single   'Probabilidade de mutação
  PElit    As Single   'Percentagem de elitismo
  Semente  As Long     'Semente para geração de números aleatórios
End Type

Type TGAParam
    PopBegin    As Integer 'Tamanho da população inicial
    PopEnd      As Integer 'Tamanho da população final
```

```
    PopStep     As Integer 'Incremento do tamanho da população
    SeedBegin   As Integer 'Valor da semente inicial
    SeedEnd     As Integer 'Valor da semente final
    SeedStep    As Integer 'Incremento da semente
    RecombBegin As Single  'Probabilidade de cruzamento inicial
    RecombEnd   As Single  'Probabilidade de cruzamento final
    RecombStep  As Single  'Incremento do cruzamento
    MutatBegin  As Single  'Probabilidade de mutação inicial
    MutatEnd    As Single  'Probabilidade de mutação final
    MutatStep   As Single  'Incremento da mutação
    Niter       As Integer 'Número da geração
    ElitBegin   As Single  'Probabilidade de elitismo inicial
    ElitEnd     As Single  'Probabilidade de elitismo final
    ElitStep    As Single  'Incremento do elitismo
End Type

Type TBlockModel
      OrigX   As Single 'Coordenada X do modelo de blocos
      OrigY   As Single 'Coordenada Y do modelo de blocos
      OrigZ   As Single 'Cota inicial do modelo de blocos
      SizeBX  As Single 'Tamanho do bloco na direção X
      SizeBY  As Single 'Tamanho do bloco na direção Y
      SizeBZ  As Single 'Tamanho do bloco na direção Z

      NRow    As Integer 'Número de linhas do modelo de blocos
      NCol    As Integer 'Número de colunas do modelo de blocos
      NLevel  As Integer 'Número de níveis do modelo de blocos
      Angle   As Integer 'Ângulo geral de talude
      Value   As Single  'Valor do bloco

      Values(22, 22, 14) As Single   'Valor do bloco lido do arquivo
ASCI
      CodMin(500)        As String   'Linha, coluna e nível dos blocos
de minério
      Extract(22, 22, 14) As Boolean 'Indica se o bloco será lavrado
ou não
End Type

Sub ShowForm()
  'Exibe interface de entrada de parâmetros
  UserFormAGC.Show
End Sub


Sub Executa_Macro()
   'Excecuta o programa de acordo com os parâmetros definidos na
interface

  'Inicializa o modelo de blocos
  Call InitModel(CSng(UserFormAGC.TextBoxOX.Text),
CSng(UserFormAGC.TextBoxOY.Text), CSng(UserFormAGC.TextBoxOZ.Text), _
                 CSng(UserFormAGC.TextBoxSX.Text),
CSng(UserFormAGC.TextBoxSY.Text), CSng(UserFormAGC.TextBoxSZ.Text), _
                 Int(UserFormAGC.TextBoxNR.Text),
Int(UserFormAGC.TextBoxNC.Text), Int(UserFormAGC.TextBoxNL.Text))
  'Lê o modelo de blocos a partir do arquivo ASCII
  Call Read_Model(UserFormAGC.TextBoxFile.Text, True, True, 0,
Int(UserFormAGC.TextBoxNR.Text), Int(UserFormAGC.TextBoxNC.Text),
Int(UserFormAGC.TextBoxNL.Text))
```

```
  Update_GAParam ' Inicializa os parâmetros iniciais e finais do AGS
  Calcula_NExecucoes
  Linha = 1

  'Atualiza planilha do modelo lido
  Update_Rows
  Format_Read_Data

  'Chama o programa principal
  BeginTime = Now
  Call Execute_Main_Program("C:\Resultados.txt")
  'UserFormAGC.StatusBarAGC.SimpleText = "Pronto"
  EndTime = Now
  'Atualiza resultados na interface
  With UserFormAGC.TextBoxResults
    .Text = .Text + "Data/Hora Início: " + CStr(BeginTime) + Chr(13)
    .Text = .Text + "Data/Hora Fim:    " + CStr(EndTime) + Chr(13)
    .Text = .Text + "Valor da Cava: " + CStr(Populacao(1).Valor) +
Chr(13)
  End With
End Sub


Sub InitModel(OX As Single, OY As Single, OZ As Single, SX As Single,
SY As Single, SZ As Single, _
              NR As Integer, NC As Integer, NL As Integer)
  'Inicializa modelo
  BlockModel.OrigX = OX
  BlockModel.OrigY = OY
  BlockModel.OrigZ = OZ
  BlockModel.SizeBX = SX
  BlockModel.SizeBY = SY
  BlockModel.SizeBZ = SZ
  BlockModel.NRow = NR
  BlockModel.NCol = NC
  BlockModel.NLevel = NL
  BlockModel.Angle = 45

End Sub

Sub Read_Model(Name As String, FillSheet As Boolean, WasteModel As
Boolean, WasteValue As Single, _
           NRowMax As Integer, NColMax As Integer, NLevelMax As
Integer)

  'Lê modelo de blocos a partir do arquivo ASCII
  Dim Aux   As String
  Dim i     As Integer
  Dim j     As Integer
  Dim k     As Integer
  Dim Value As Single
  Dim NLine As Integer
  Dim fs    As Object
  Dim F     As Object

  Const ForReading = 1, ForWriting = 2, ForAppending = 3
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set F = fs.OpenTextFile(Name, ForReading)
```

```
  'Le o cabeçaho
  Aux = F.ReadLine
  'Leitura dos valores dos blocos
  While Not F.AtEndOfStream
    Aux = F.ReadLine
    Call Read_IJK_Values(Aux, i, j, k, Value)
    BlockModel.Values(i, j, k) = Value
    If (Not WasteModel) And (Value < 0) Then BlockModel.Values(i, j,
k) = WasteValue
  Wend

  F.Close

  'Atualiza planilha de acordo com o arquivo ASCII
  NLine = 1
  Comp_Cromossomo = 0

  For i = 1 To NRowMax
    For j = 1 To NColMax
      For k = 1 To NLevelMax
        NLine = NLine + 1
        BlockModel.Extract(i, j, k) = False
        If BlockModel.Values(i, j, k) >= 0 Then
          Comp_Cromossomo = Comp_Cromossomo + 1
          BlockModel.CodMin(Comp_Cromossomo) = IJK_to_string(i, j, k)
        End If
      Next k
    Next j
  Next i

  'Limpa planilha
  Clear_Sheet ("Model_CSV")

  If FillSheet Then
    NLine = 0
    For i = 1 To NRowMax
      For j = 1 To NColMax
        For k = 1 To NLevelMax
          NLine = NLine + 1
          Worksheets("Model_CSV").Cells(NLine, 1).Value = i
          Worksheets("Model_CSV").Cells(NLine, 2).Value = j
          Worksheets("Model_CSV").Cells(NLine, 3).Value = k
          Worksheets("Model_CSV").Cells(NLine, 4).Value =
BlockModel.Values(i, j, k)
        Next k
      Next j
    Next i
  End If
End Sub

Sub Read_IJK_Values(Aux As String, i As Integer, j As Integer, k As
Integer, Value As Single)

  'Converte a variavel Aux (linha) para os indices e valores do modelo
de blocos
  Dim Begin_String(10) As Integer
  Dim Len_String(10)   As Integer
  Dim Index            As Integer
```

```
  Dim z                As Integer

  Index = 1
  Begin_String(0) = 0
  Len_String(0) = 0
  Begin_String(Index) = 1
  For z = 1 To Len(Aux)
    If (Mid(Aux, z, 1) = ",") Then
      Len_String(Index) = z - 1 - Len_String(Index - 1) -
Begin_String(Index - 1)
      Index = Index + 1
      Begin_String(Index) = z + 1
    End If
  Next z
  Len_String(Index) = Len(Aux) - Begin_String(Index - 1) - 1

  i = Int(Mid(Aux, Begin_String(1), Len_String(1)))
  j = Int(Mid(Aux, Begin_String(2), Len_String(2)))
  k = Int(Mid(Aux, Begin_String(3), Len_String(3)))
  Value = CDbl(Mid(Aux, Begin_String(4), Len_String(4)))
End Sub

Sub Execute_Main_Program(FileResult As String)

' Executa o AGS tantas vezes quanto forem os parâmetros iniciais
definidos
  Dim fs As Object
  Dim F  As Object
  Dim i  As Integer

  Set fs = CreateObject("Scripting.FileSystemObject")
  Set F = fs.CreateTextFile(FileResult)


  ' Inicializa arquivo de resultados
  F.WriteLine ("Sem.,Pop.,Cruz.,Mut,Elit,Iter.,Valor")

  With AGS
    .TamPop = GAParam.PopBegin               'Tamanho da população
    .GerMax = GAParam.Niter                  'Número máximo de
gerações
    .Semente = GAParam.SeedBegin           'Semente
    .PCruz = GAParam.RecombBegin           'Probabilidade de
cruzamento
    .PMut = GAParam.MutatBegin             'Probabilidade de mutação
    .PElit = GAParam.ElitBegin             'Percentagem de elitismo
  End With
  While AGS.Semente <= GAParam.SeedEnd
    While AGS.TamPop <= GAParam.PopEnd
      While AGS.PCruz <= GAParam.RecombEnd
        While AGS.PMut <= GAParam.MutatEnd
          While AGS.PElit <= GAParam.ElitEnd
            Executa_AGS
            EscreveResultados
            Linha = Linha + 1
            F.WriteLine (Str(AGS.Semente) + "," + Str(AGS.TamPop) +
"," + Str(AGS.PCruz) + "," + _
              Str(AGS.PMut) + "," + Str(AGS.PElit) + "," +
Str(GAParam.Niter) + "," + _
```

```
              Str(Populacao(1).Valor))
            AGS.PElit = AGS.PElit + GAParam.ElitStep
          Wend
          AGS.PElit = GAParam.ElitBegin
          AGS.PMut = AGS.PMut + GAParam.MutatStep
        Wend
        AGS.PElit = GAParam.ElitBegin
        AGS.PMut = GAParam.MutatBegin
        AGS.PCruz = AGS.PCruz + GAParam.RecombStep
      Wend
      AGS.PElit = GAParam.ElitBegin
      AGS.PMut = GAParam.MutatBegin
      AGS.PCruz = GAParam.RecombBegin
      AGS.TamPop = AGS.TamPop + GAParam.PopStep
    Wend
    AGS.PElit = GAParam.ElitBegin
    AGS.PMut = GAParam.MutatBegin
    AGS.PCruz = GAParam.RecombBegin
    AGS.TamPop = GAParam.PopBegin
    AGS.Semente = AGS.Semente + GAParam.SeedStep
  Wend
  F.Close

  For i = 1 To Comp_Cromossomo
    'Marca blocos que necessitam ser extraídos do modelo de blocos
(estéril+minério)
    If Populacao(1).Cromossomo(i) Then
      Call Extract_Block(Int(Mid(BlockModel.CodMin(i), 1, 3)),
Int(Mid(BlockModel.CodMin(i), 4, 3)), _
                    Int(Mid(BlockModel.CodMin(i), 7, 3)))
    End If
  Next i

  Update_Extract_Rows
  Format_Extract_Data
End Sub

Sub Update_GAParam()
' Atualiza Parâmetros do AGS de acordo com a interface
  With GAParam
    .PopBegin = Int(UserFormAGC.TextBoxPopB.Text)
    If Not UserFormAGC.CheckBoxPop.Value Then
      .PopEnd = Int(UserFormAGC.TextBoxPopE.Text)
    Else
      .PopEnd = .PopBegin
    End If
    .PopStep = Int(UserFormAGC.TextBoxPopS.Text)
    .SeedBegin = Int(UserFormAGC.TextBoxSeedB.Text)
    If Not UserFormAGC.CheckBoxSeed.Value Then
      .SeedEnd = Int(UserFormAGC.TextBoxSeedE.Text)
    Else
      .SeedEnd = .SeedBegin
    End If
    .SeedStep = Int(UserFormAGC.TextBoxSeedS.Text)
    .RecombBegin = CSng(UserFormAGC.TextBoxRecB.Text)
    If Not UserFormAGC.CheckBoxRec.Value Then
      .RecombEnd = CSng(UserFormAGC.TextBoxRecE.Text)
    Else
      .RecombEnd = .RecombBegin
```

```
    End If
    .RecombStep = CSng(UserFormAGC.TextBoxRecS.Text)

    .MutatBegin = CSng(UserFormAGC.TextBoxMutB.Text)
    If Not UserFormAGC.CheckBoxMut.Value Then
      .MutatEnd = CSng(UserFormAGC.TextBoxMutE.Text)
    Else
      .MutatEnd = .MutatBegin
    End If
    .MutatStep = CSng(UserFormAGC.TextBoxMutS.Text)

    .ElitBegin = CSng(UserFormAGC.TextBoxElitB.Text)
    If Not UserFormAGC.CheckBoxElit.Value Then
      .ElitEnd = CSng(UserFormAGC.TextBoxElitE.Text)
    Else
      .ElitEnd = .ElitBegin
    End If
    .ElitStep = CSng(UserFormAGC.TextBoxElitS.Text)
    .Niter = Int(UserFormAGC.TextBoxIter.Text)
  End With
End Sub

Sub Executa_AGS()
  'Programa principal do Algoritmo Genético Simples (AGS) com elitismo
  Dim i        As Integer

  Inicializa_Populacao  'Gera população aleatória inicial
  Niter = 0             'Variável de controle do número de iterações
(gerações)
  While (Niter < AGS.GerMax)
    Niter = Niter + 1
    If Niter = 1 And Linha = NExec Then Clear_Sheet ("Resultados")
    For i = 1 To AGS.TamPop
      Call Avalia_Individuo(i)  'Calcula valores da populacao segundo
a funcao original
    Next i
    Call Seleciona_MelhorInd             'Coloca na posição 1 o melhor
individuo
    'UserFormAGC.StatusBarAGC.SimpleText = "Calculando... Geração [" +
CStr(Niter) + "] de " + CStr(AGS.GerMax) + _
    '  "  -  Pit Value: " + CStr(Populacao(1).Valor)
    Reproducao             'Seleciona os indivíduos da próxima geração
    Cruzamento             'Efetua o cruzamento de acordo com a pc
    Mutacao                'Efetua a mutação de acordo com a pm
  Wend
  For i = 1 To AGS.TamPop
      Call Avalia_Individuo(i)  'Calcula valores da populacao segundo
a funcao original
  Next i
  Call Seleciona_MelhorInd 'Coloca na posição 1 o melhor individuo
End Sub

Sub Inicializa_Populacao()
  Dim i As Integer
  Dim j As Integer

  If Linha = NExec Then
    Clear_Sheet ("PopIni")
    Clear_Sheet ("PopFit")
```

```
  End If

  Randomize (AGS.Semente) 'Reseta a função de geração de número
aleatório
  'Inicializa cromossomos e calcula valores dos parâmetros da função
  For i = 1 To AGS.TamPop
    For j = 1 To Comp_Cromossomo
      If Round(Rnd) = 1 Then
        Populacao(i).Cromossomo(j) = True
      Else
        Populacao(i).Cromossomo(j) = False
      End If
      If Linha = NExec Then Worksheets("PopIni").Cells(j, i).Value =
Populacao(i).Cromossomo(j)
    Next j
  Next i
End Sub

Sub Avalia_Individuo(Indiv As Integer)
  Dim i As Integer

  For i = 1 To Comp_Cromossomo
    'Marca blocos que necessitam ser extraídos do modelo de blocos
(estéril+minério)
    If Populacao(Indiv).Cromossomo(i) Then
      Call Extract_Block(Int(Mid(BlockModel.CodMin(i), 1, 3)),
Int(Mid(BlockModel.CodMin(i), 4, 3)), _
                  Int(Mid(BlockModel.CodMin(i), 7, 3)))
    End If
  Next i
  Call Pit_Value(Indiv)

  If Linha = NExec Then Worksheets("Resultados").Cells(Niter,
Indiv).Value = Populacao(Indiv).Valor

End Sub

Sub Pit_Value(Indiv As Integer)
  'Atualiza o valor da cava de acordo com os blocos extraídos
  Dim i As Integer
  Dim j As Integer
  Dim k As Integer
  BlockModel.Value = 0
  For i = 1 To BlockModel.NRow
    For j = 1 To BlockModel.NCol
      For k = 1 To BlockModel.NLevel
        If BlockModel.Extract(i, j, k) Then
          BlockModel.Value = BlockModel.Value + BlockModel.Values(i,
j, k)
          BlockModel.Extract(i, j, k) = False
        End If
      Next k
    Next j
  Next i
  Populacao(Indiv).Valor = BlockModel.Value
End Sub

Sub Seleciona_MelhorInd()
  Dim i      As Integer
```

```
  Dim j      As Integer
  Dim VMax   As Single
  Dim Vold   As Single
  Dim Vindex As Integer

  'Procura mellhor indivíduo na populacao baseado no valores do modelo
de blocos
  VMax = -1E+32
  Lamb = 1E+32
  For i = 1 To AGS.TamPop
    If Populacao(i).Valor > VMax Then
      Vindex = i
      VMax = Populacao(i).Valor
    End If
    'Determina o valor de lambida segundo o menor valor da função
    If Populacao(i).Valor < Lamb Then Lamb = Populacao(i).Valor
  Next i

  Lamb = Abs(Lamb) + 0.001

  If Vindex <> 1 Then
    'Preserva valores antigos do primeiro individuo
    For i = 1 To Comp_Cromossomo
      Cold(i) = Populacao(1).Cromossomo(i)
    Next i
    Vold = Populacao(1).Valor
    'Transfere o melhor individuo para a primeira posição da população
    For i = 1 To Comp_Cromossomo
      Populacao(1).Cromossomo(i) = Populacao(Vindex).Cromossomo(i)
    Next i
    Populacao(1).Valor = Populacao(Vindex).Valor
    For i = 1 To Comp_Cromossomo
      Populacao(Vindex).Cromossomo(i) = Cold(i)
    Next i
    Populacao(Vindex).Valor = Vold
  End If

  ' Copia melhor individuo de acordo com ao percentual de elitismo
  For i = 2 To Round(AGS.PElit * AGS.TamPop) - 1
    For j = 1 To Comp_Cromossomo
      Populacao(i).Cromossomo(j) = Populacao(1).Cromossomo(j)
    Next j
    Populacao(i).Valor = Populacao(1).Valor
  Next i
End Sub

Sub Reproducao()
  Dim i          As Integer
  Dim j          As Integer
  Dim AcumAp     As Single  'Somatório da função de mérito
  Dim AcumProb   As Single  'Somatório da probabilidade
  Dim NC         As Integer 'Número de cópias do indivíduo
  Dim NAleatorio As Single  'Número aleatório entre 0 e 1
  Dim NCmax      As Integer 'Número de cópias máxima da população
  Dim IndMax     As Integer 'Indivíduo que possui o maior número de
cópias

  'Calcula a função de mérito da população
  AcumAp = 0
```

```
  For i = 1 To AGS.TamPop
    Populacao(i).Aptidao = Populacao(i).Valor + Lamb
    If Linha = NExec Then Worksheets("PopFit").Cells(Niter, i).Value =
Populacao(i).Aptidao
    AcumAp = AcumAp + Populacao(i).Aptidao
  Next i
  AcumProb = 0
  'Calcula a probabilidade de cada individuo da população
  For i = 1 To AGS.TamPop
    ProbPop(i) = Populacao(i).Aptidao / AcumAp
    AcumProb = AcumProb + ProbPop(i)
    NCopias(i) = 0
    ProbPopAc(i) = AcumProb
  Next i

  'Calcula o número de cópias de cada indivíduo baseado no método da
roleta
  NC = 0
  For i = 1 To AGS.TamPop
    NAleatorio = Rnd
    For j = 1 To AGS.TamPop - 1
      If (NAleatorio >= ProbPopAc(j)) And (NAleatorio < ProbPopAc(j +
1)) Then
        NCopias(j) = NCopias(j) + 1
        NC = NC + 1
      End If
    Next j
  Next i
  If NC < AGS.TamPop Then
    For i = NC + 1 To AGS.TamPop
      NCopias(AGS.TamPop) = NCopias(AGS.TamPop) + 1
    Next i
  End If

  'Caso o melhor individuo não seja selecionado faz-se a seleção deste
e decrementa-se
  'do individuo com o menor número de cópias
  NCmax = 0
  If NCopias(1) = 0 Then
    NCopias(1) = 1
    For i = 2 To AGS.TamPop
      If NCmax < NCopias(i) Then
        NCmax = NCopias(i)
        IndMax = i
      End If
    Next i
    NCopias(IndMax) = NCopias(IndMax) - 1
  End If

  'Escreve na planilha o número de cópias da população
  If Niter = 1 And Linha = NExec Then Clear_Sheet ("NCopias")
  NC = 0
  For i = 1 To AGS.TamPop
    NC = NC + NCopias(i)
    If Linha = NExec Then Worksheets("NCopias").Cells(Niter, i).Value
= NCopias(i)
  Next i
  If Linha = NExec Then Worksheets("NCopias").Cells(Niter, AGS.TamPop
+ 1).Value = NC
```

```
End Sub

Sub Cruzamento()
  Dim i          As Integer
  Dim j          As Integer
  Dim k          As Integer
  Dim NC         As Integer
  Dim NAleatorio As Single
  'Gera a nova população baseado na seleção da variável "NCopias"
  NC = 0
  For i = 1 To AGS.TamPop
    If NCopias(i) > 0 Then
      For j = 1 To NCopias(i)
        NC = NC + 1
        For k = 1 To Comp_Cromossomo
          CopiaCromos(NC, k) = Populacao(i).Cromossomo(k)
        Next k
      Next j
    End If
  Next i

  'Executa cruzamento e atualiza população
  i = 1
  While i <= (AGS.TamPop - 2)
    NAleatorio = Rnd
    If NAleatorio <= AGS.PCruz Then
      Call Crossover(i + 1)
    Else
      For j = 1 To Comp_Cromossomo
        Populacao(i + 1).Cromossomo(j) = CopiaCromos(i + 1, j)
        Populacao(i + 2).Cromossomo(j) = CopiaCromos(i + 2, j)
      Next j
    End If
    i = i + 2
  Wend

  'Escreve cópia da populacao inicial
  If Niter = 1 Then
    If Linha = NExec Then Clear_Sheet ("PopCopy")
    For i = 1 To AGS.TamPop
      For j = 1 To Comp_Cromossomo
        If Linha = NExec Then Worksheets("PopCopy").Cells(j, i).Value
= CopiaCromos(i, j)
      Next j
    Next i
  End If
End Sub

Sub Crossover(IndPop As Integer)
  'Executa cruzamento de um ponto
  Dim i          As Integer
  Dim Fim        As Integer
  Dim NAleatorio As Integer
  NAleatorio = Round((Comp_Cromossomo - 1 + 1) * Rnd + 1)
  For i = 1 To (NAleatorio - 1)
    Populacao(IndPop).Cromossomo(i) = CopiaCromos(IndPop, i)
    Populacao(IndPop + 1).Cromossomo(i) = CopiaCromos(IndPop + 1, i)
  Next i
  Fim = Comp_Cromossomo
```

```
  For i = NAleatorio To Fim
    Populacao(IndPop).Cromossomo(i) = CopiaCromos(IndPop + 1, i)
    Populacao(IndPop + 1).Cromossomo(i) = CopiaCromos(IndPop, i)
  Next i
End Sub

Sub Mutacao()
  Dim i          As Integer
  Dim j          As Integer
  Dim NAleatorio As Single

  For i = 2 To AGS.TamPop
    For j = 1 To Comp_Cromossomo
      NAleatorio = Rnd
      If NAleatorio <= AGS.PMut Then
        If Populacao(i).Cromossomo(j) = True Then
          Populacao(i).Cromossomo(j) = False
        Else
          Populacao(i).Cromossomo(j) = True
        End If
      End If
    Next j
  Next i
End Sub

Sub EscreveResultados()
  If Linha = 1 Then Clear_Sheet ("Resultados_Grupo")
  Sheets("Resultados_Grupo").Activate
  With Worksheets("Resultados_Grupo")
    Cells(Linha, 1).Value = Linha
    Cells(Linha, 2).Value = AGS.Semente
    Cells(Linha, 3).Value = AGS.TamPop
    Cells(Linha, 4).Value = AGS.PCruz
    Cells(Linha, 5).Value = AGS.PMut
    Cells(Linha, 6).Value = AGS.PElit
    Cells(Linha, 7).Value = GAParam.Niter
    Cells(Linha, 8).Value = Populacao(1).Valor
  End With
End Sub


Function IJK_to_string(i As Integer, j As Integer, k As Integer) As
String
  Dim si As String
  Dim sj As String
  Dim sk As String

  si = Insert_Zero_Left(i)
  sj = Insert_Zero_Left(j)
  sk = Insert_Zero_Left(k)
  IJK_to_string = Trim(Trim(si) + Trim(sj) + Trim(sk))
End Function

Function Insert_Zero_Left(N As Integer) As String
  If N <= 9 Then
    Insert_Zero_Left = Trim("00" + Trim(Str(N)))
  Else
    If N <= 99 Then
      Insert_Zero_Left = Trim("0" + Trim(Str(N)))
```

```
    Else
      Insert_Zero_Left = Trim(Str(N))
    End If
  End If
End Function

Sub Extract_Block(i As Integer, j As Integer, k As Integer)
  'Lavra os blocos de acordo com índices  ijk e do angulo geral de
talude
  Dim L As Integer
  Dim N As Integer

  Call Extract_Block_Above(i, j, k)

  N = 1
  For L = k - 1 To 1 Step -1
    Call Extract_Block_Around(i, j, L, N)
    N = N + 1
  Next L
End Sub

Sub Extract_Block_Around(i As Integer, j As Integer, k As Integer, N
As Integer)
  'Lavra os blocos de acordo com o ângulo de talude
  Dim TanAngle As Single
  Dim Pi       As Single
  Dim iRow     As Integer
  Dim iRowMax  As Integer
  Dim iRowMin  As Integer
  Dim iCol     As Integer
  Dim iColMax  As Integer
  Dim iColMin  As Integer
  Dim Aux      As Single

  Pi = 3.14159265358979
  TanAngle = Tan(BlockModel.Angle * Pi / 180)

  iRowMax = i + (BlockModel.SizeBZ / BlockModel.SizeBY + 1) * N
  If iRowMax > BlockModel.NRow Then iRowMax = BlockModel.NRow
  iRowMin = i - (BlockModel.SizeBZ / BlockModel.SizeBY + 1) * N
  If iRowMin < 1 Then iRowMin = 1
  iColMax = j + (BlockModel.SizeBZ / BlockModel.SizeBX + 1) * N
  If iColMax > BlockModel.NCol Then iColMax = BlockModel.NCol
  iColMin = j - (BlockModel.SizeBZ / BlockModel.SizeBX + 1) * N
  If iColMin < 1 Then iColMin = 1

  For iRow = iRowMin To iRowMax
    For iCol = iColMin To iColMax
      Aux = Sqr((i - iRow) * BlockModel.SizeBY * (i - iRow) *
BlockModel.SizeBY + (j - iCol) * BlockModel.SizeBX * (j - iCol) *
BlockModel.SizeBX)
      If (Aux <= (BlockModel.SizeBZ * N / TanAngle)) Then
        BlockModel.Extract(iRow, iCol, k) = True
      End If
    Next iCol
  Next iRow
End Sub

Sub Extract_Block_Above(i As Integer, j As Integer, k As Integer)
```

```
  'Lavra os blocos acima do bloco atual
  Dim L As Integer
  If (i <= BlockModel.NRow) And (j <= BlockModel.NCol) And (k <=
BlockModel.NLevel) And (i > 0) And (j > 0) Then
    For L = 1 To k
      BlockModel.Extract(i, j, L) = True
    Next L
  End If
End Sub
Sub Clear_Sheet(Name As String)
  'Apaga a planilha repassada em Name
  Dim NewSheet As Worksheet
  Set NewSheet = Worksheets.Add
  NewSheet.Name = Name

  With Worksheets(Name)
    Sheets(Name).Select
    Sheets(Name).Activate
    Cells.Select
    Selection.ClearContents
    Range("A1").Select
    ActiveWindow.DisplayGridlines = False
  End With
End Sub

Sub Update_Rows()
  'Atualiza as planilhas da linhas do modelo de blocos
  Dim i As Integer
  Dim j As Integer
  Dim k As Integer

  'Apaga dados existentes
  For i = 1 To BlockModel.NRow
    Clear_Sheet (Trim("r" + Trim(Str(i))))
  Next i

  'Atualiza planilha de acordo com o modelo de blocos
  For i = 1 To BlockModel.NRow
    For j = 1 To BlockModel.NCol
      For k = 1 To BlockModel.NLevel
        Worksheets("r" + Trim(Str(i))).Cells(k, j).Value =
BlockModel.Values(i, j, k)
      Next k
    Next j
  Next i
End Sub
Sub Format_Read_Data()
  'Formata os dados lidos da linhas
  Dim i    As Integer
  Dim Name As String

  For i = 1 To BlockModel.NRow
    Name = Trim("r" + Trim(Str(i)))
    Format_Read_Data_Color (Name)
  Next i
End Sub
Sub Format_Read_Data_Color(Name As String)
  With Sheets(Name)
    .Activate
```

```
    Range("A1").Select
    Range(Selection, ActiveCell.SpecialCells(xlLastCell)).Select

    'Desenha as linhas de contorno das células
    Draw_Gridlines

    'Ajusta autmaticamnete altura e largura da planilha
    Cells.EntireColumn.AutoFit

    'Define as cores
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions.Delete
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions.Add
Type:=xlCellValue, Operator:=xlBetween, _
        Formula1:="-9999999", Formula2:="-0.001"
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions(1).Interior.Colo
rIndex = 36
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions.Add
Type:=xlCellValue, Operator:=xlBetween, _
        Formula1:="0", Formula2:="8"
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions(2).Interior.Colo
rIndex = 8
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions.Add
Type:=xlCellValue, Operator:=xlBetween, _
        Formula1:="8.001", Formula2:="99999999"
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions(3).Interior.Colo
rIndex = 3
    Range("A1").Select
  End With
End Sub

Sub Update_Extract_Rows()
  'Atualiza planilha dos blocos lavrados
  Dim i As Integer
  Dim j As Integer
  Dim k As Integer

  'Apaga dados atuais
  For i = 1 To BlockModel.NRow
    Clear_Sheet (Trim("Er" + Trim(Str(i))))
  Next i

  For i = 1 To BlockModel.NRow
    For j = 1 To BlockModel.NCol
      For k = 1 To BlockModel.NLevel
        If BlockModel.Extract(i, j, k) Then
          Worksheets("Er" + Trim(Str(i))).Cells(k, j).Value = True
        Else
          Worksheets("Er" + Trim(Str(i))).Cells(k, j).Value = False
        End If
      Next k
    Next j
  Next i
```

```
End Sub

Sub Format_Extract_Data()
  'Formata planilha dos blocos lavrados
  Dim i    As Integer
  Dim Name As String
  For i = 1 To BlockModel.NRow
      Name = Trim("Er" + Trim(Str(i)))
    Format_Extract_Data_Color (Name)
  Next i
End Sub

Sub Format_Extract_Data_Color(Name As String)
  'Formata planilha dos blocos lavrados
  With Sheets(Name)
    .Activate
    Range("A1").Select
    Range(Selection, ActiveCell.SpecialCells(xlLastCell)).Select

    'Draw Gridlines
    Draw_Gridlines

    'Adjust width and height cells automatic
    Cells.EntireColumn.AutoFit

    'Set colors
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions.Delete
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions.Add
Type:=xlCellValue, _
      Operator:=xlEqual, Formula1:="TRUE"
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions(1).Interior.Colo
rIndex = 35
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions.Add
Type:=xlCellValue, _
      Operator:=xlEqual, Formula1:="FALSE"
    Range(Selection,
ActiveCell.SpecialCells(xlLastCell)).FormatConditions(2).Interior.Colo
rIndex = 36
    Range("A1").Select
  End With
End Sub

Sub Draw_Gridlines()
    'Desenha contornos nas células
    Selection.Borders(xlDiagonalDown).LineStyle = xlNone
    Selection.Borders(xlDiagonalUp).LineStyle = xlNone
    With Selection.Borders(xlEdgeLeft)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeTop)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
```

```
    End With
    With Selection.Borders(xlEdgeBottom)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeRight)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlInsideVertical)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    If BlockModel.NRow > 1 Then
      With Selection.Borders(xlInsideHorizontal)
          .LineStyle = xlContinuous
          .Weight = xlThin
          .ColorIndex = xlAutomatic
      End With
    End If
End Sub

Sub Calcula_NExecucoes()
  Dim NIPop  As Integer
  Dim NIRec  As Integer
  Dim NIMut  As Integer
  Dim NIElit As Integer
  Dim NISeed As Integer

  With UserFormAGC
    If .CheckBoxPop.Value Then
      NIPop = 1
    Else
      NIPop = Round(GAParam.PopEnd - GAParam.PopBegin) /
(GAParam.PopStep + 1)
    End If
    If .CheckBoxRec.Value Then
      NIRec = 1
    Else
      NIRec = Round(GAParam.RecombEnd - GAParam.RecombBegin) /
(GAParam.RecombStep + 1)
    End If
    If .CheckBoxMut.Value Then
      NIMut = 1
    Else
      NIMut = Round(GAParam.MutatEnd - GAParam.MutatBegin) /
(GAParam.MutatStep + 1)
    End If
    If .CheckBoxSeed.Value Then
      NISeed = 1
    Else
      NISeed = Round(GAParam.SeedEnd - GAParam.SeedBegin) /
(GAParam.SeedStep + 1)
    End If
    If .CheckBoxElit.Value Then
      NIElit = 1
```

```
    Else
      NIElit = Round(GAParam.ElitEnd - GAParam.ElitBegin) /
(GAParam.ElitStep + 1)
    End If
  End With
  NExec = Int(NIPop * NIRec * NIMut * NISeed * NIElit)
End Sub
```

8.2 Anexo II - Listagem do programa multiobjetivo NSGA II com elitismo

Baseado em Deb et al. (2000).

```
Option Explicit

Public MPop()       As Byte     ' Matriz mxn com população de
indivíduos gerados (binário) indiv=col
Public Lvar(2, 2)   As Single   ' Vetor com os limites das variáveis
Public poptemp()    As Single   '
Public MIDelit()    As Single   ' População externa para o elitismo
Public FxI()        As Single   ' Matriz dos valores das funcões
(Linhas são os individuos na FxI)
Public avalia()     As Single   '
Public AVAelit()    As Single   '
Public nvar         As Integer  '
Public nfunc        As Integer  ' Número de funções objetivo
Public Nbpop        As Integer  ' Tamanho da população de indivíduos
Public NbMIDelit(2) As Integer  ' Variável de controle do tamanho da
população externa para o elitismo 1 inicio, 2 fim
Public rank()       As Integer  '
Public ja           As Integer  '
Public shar()       As Single   '
Public garank()     As Single   '
Public F1()         As Single   '
Public F2()         As Single   '
Public X1()         As Single   '
Public X2()         As Single   '
Public MaxF()       As Single   '
Public MinF()       As Single   '
Public MaxX()       As Single   '
Public MinX()       As Single   '
Public FatPar       As Single   ' Fator de partilha
Public PNS          As Single   ' Percentagem de elitismo
Public Vmul()       As Integer
Public ngen         As Integer  ' Número de genes do indivíduo

Public BlockModel  As TBlockModel    ' Parâmetros do modelo de blocos
Public AGS         As TAGS           ' Contem os parâmetros do AGS
Public nger        As Integer        ' Número de gerações
Public Pond(2)     As Single         ' Ponderadores a serem utilizados
na função monoobjetivo


Type TAGS
  TamPop   As Integer  'Tamanho da população
  GerMax   As Integer  'Numero de gerações
  Precisao As Single   'Precisão
  PCruz    As Single   'Probabilidade de cruzamento
  PMut     As Single   'Probabilidade de mutação
  PElit    As Single   'Percentagem de elitismo
  Semente  As Long     'Semente para geração de números aleatórios
End Type

Type TBlockModel
      OrigX   As Single 'Coordenada X do modelo de blocos
      OrigY   As Single 'Coordenada Y do modelo de blocos
      OrigZ   As Single 'Cota inicial do modelo de blocos
      SizeBX  As Single 'Tamanho do bloco na direção X
      SizeBY  As Single 'Tamanho do bloco na direção Y
      SizeBZ  As Single 'Tamanho do bloco na direção Z

      NRow    As Integer 'Número de linhas do modelo de blocos
```

```
      NCol    As Integer 'Número de colunas do modelo de blocos
      NLevel  As Integer 'Número de níveis do modelo de blocos
      Angle   As Integer 'Ângulo geral de talude
      Value   As Single  'Valor da cava
      Mass    As Single  'Massa da cava

      Values(22, 22, 14) As Single   'Valor do bloco lido do arquivo
ASCI
      CodMin(792)        As String   'Linha, coluna e nível dos blocos
de minério
      Extract(22, 22, 14) As Boolean 'Indica se o bloco será lavrado
ou não
End Type

Sub Executa_AG_Multi_Mono()
    'Inicializa o modelo de blocos
    Call InitModel
    'Lê o modelo de blocos a partir do arquivo ASCII
    Call Read_Model(Worksheets("Parametros").Cells(14, 2).Value,
False, True, 0, _
                    BlockModel.NRow, BlockModel.NCol,
BlockModel.NLevel)

    ' Executa o AG NSGA para os dois problemas propostos
    Worksheets("Parametros").Cells(5, 2).Value = 1
    Call Programa_MNSGA
    Worksheets("Parametros").Cells(5, 2).Value = 2
    Call Programa_MNSGA
    MsgBox "Processamento concluído para os dois problemas!"
End Sub
Sub Executa_AG_Selecionado()
    'Inicializa o modelo de blocos
    Call InitModel
    'Lê o modelo de blocos a partir do arquivo ASCII
    Call Read_Model(Worksheets("Parametros").Cells(14, 2).Value,
False, True, 0, _
                    BlockModel.NRow, BlockModel.NCol,
BlockModel.NLevel)

    ' Executa o AG NSGA para o problema selecionado
    Call Programa_MNSGA
    MsgBox "Processamento concluído para o problema selecionado!"
End Sub
Sub InitModel()
  'Inicializa modelo
  BlockModel.OrigX = -5
  BlockModel.OrigY = -5
  BlockModel.OrigZ = 0
  BlockModel.SizeBX = 10
  BlockModel.SizeBY = 10
  BlockModel.SizeBZ = 10
  BlockModel.NRow = Worksheets("Parametros").Cells(13, 2).Value
  BlockModel.NCol = 22
  BlockModel.NLevel = 14
  BlockModel.Angle = 45
  Pond(1) = Worksheets("Parametros").Cells(11, 2).Value
  Pond(2) = Worksheets("Parametros").Cells(12, 2).Value
End Sub
```

```
Sub Read_Model(Name As String, FillSheet As Boolean, WasteModel As
Boolean, WasteValue As Single, _
           NRowMax As Integer, NColMax As Integer, NLevelMax As
Integer)

  'Lê modelo de blocos a partir do arquivo ASCII
  Dim Aux   As String
  Dim i     As Integer
  Dim j     As Integer
  Dim k     As Integer
  Dim Value As Single
  Dim NLine As Integer
  Dim fs    As Object
  Dim F     As Object

  Const ForReading = 1, ForWriting = 2, ForAppending = 3
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set F = fs.OpenTextFile(Name, ForReading)

  'Le o cabeçaho
  Aux = F.ReadLine
  'Leitura dos valores dos blocos
  While Not F.AtEndOfStream
    Aux = F.ReadLine
    Call Read_IJK_Values(Aux, i, j, k, Value)
    BlockModel.Values(i, j, k) = Value
    If (Not WasteModel) And (Value < 0) Then BlockModel.Values(i, j,
k) = WasteValue
  Wend

  F.Close

  'Atualiza planilha de acordo com o arquivo ASCII
  NLine = 1
  ngen = 0

  For i = 1 To NRowMax
    For j = 1 To NColMax
      For k = 1 To NLevelMax
        NLine = NLine + 1
        BlockModel.Extract(i, j, k) = False
        If BlockModel.Values(i, j, k) >= 0 Then
          ngen = ngen + 1
          BlockModel.CodMin(ngen) = IJK_to_string(i, j, k)
        End If
      Next k
    Next j
  Next i

  'Limpa planilha
  'Clear_Sheet ("Model_CSV")

  If FillSheet Then
    NLine = 0
    For i = 1 To NRowMax
      For j = 1 To NColMax
        For k = 1 To NLevelMax
          NLine = NLine + 1
          Worksheets("Model_CSV").Cells(NLine, 1).Value = i
```

```
          Worksheets("Model_CSV").Cells(NLine, 2).Value = j
          Worksheets("Model_CSV").Cells(NLine, 3).Value = k
          Worksheets("Model_CSV").Cells(NLine, 4).Value =
BlockModel.Values(i, j, k)
        Next k
      Next j
    Next i
  End If
End Sub

Sub Read_IJK_Values(Aux As String, i As Integer, j As Integer, k As
Integer, Value As Single)

  'Converte a variavel Aux (linha) para os indices e valores do modelo
de blocos
  Dim Begin_String(10) As Integer
  Dim Len_String(10)   As Integer
  Dim Index            As Integer
  Dim z                As Integer

  Index = 1
  Begin_String(0) = 0
  Len_String(0) = 0
  Begin_String(Index) = 1
  For z = 1 To Len(Aux)
    If (MID(Aux, z, 1) = ",") Then
      Len_String(Index) = z - 1 - Len_String(Index - 1) -
Begin_String(Index - 1)
      Index = Index + 1
      Begin_String(Index) = z + 1
    End If
  Next z
  Len_String(Index) = Len(Aux) - Begin_String(Index - 1) - 1

  i = Int(MID(Aux, Begin_String(1), Len_String(1)))
  j = Int(MID(Aux, Begin_String(2), Len_String(2)))
  k = Int(MID(Aux, Begin_String(3), Len_String(3)))
  Value = CSng(MID(Aux, Begin_String(4), Len_String(4)))
End Sub
Function IJK_to_string(i As Integer, j As Integer, k As Integer) As
String
  Dim si As String
  Dim sj As String
  Dim sk As String

  si = Insert_Zero_Left(i)
  sj = Insert_Zero_Left(j)
  sk = Insert_Zero_Left(k)
  IJK_to_string = Trim(Trim(si) + Trim(sj) + Trim(sk))
End Function

Function Insert_Zero_Left(N As Integer) As String
  If N <= 9 Then
    Insert_Zero_Left = Trim("00" + Trim(Str(N)))
  Else
    If N <= 99 Then
      Insert_Zero_Left = Trim("0" + Trim(Str(N)))
    Else
      Insert_Zero_Left = Trim(Str(N))
```

```
    End If
  End If
End Function
Sub Programa_MNSGA()
    ' Inicialização de Variáveis
    Dim Nbgen     As Integer ' Número de gerações
    Dim Nopt      As Integer ' Número de variáveis de otimização
(cromossomos)
    Dim Nbit      As Integer ' Tamanho do cromossomo (número de bits)
    Dim Pcros     As Single  ' Probabilidade de cruzamento
    Dim PMut      As Single  ' Probabilidade de mutação de cada bit
(gene)
    Dim adapt     As Integer ' Entrada do tipo de adaptação
    Dim Iselec    As Integer ' Entrada do tipo de seleção
    Dim FI()      As Single  ' Vetor de desempenho gerado pelo NSGA
    Dim M()       As Single  ' Matriz com as variáveis de otimização
    Dim G         As Single  ' Fator g
    Dim NPext     As Integer ' Tamanho máximo da população externa
    Dim NOVO      As Integer ' Tamanho da população externa
    Dim Ntype     As Integer ' Tipo de problema
    Dim Tcruz     As Integer ' Tipo de cruzamento
    Dim NSel      As Integer ' Número de indivíduos as serem
processados
    Dim F         As Integer
    Dim Dt        As Single
    Dim DrI()     As Single
    Dim DrIM()    As Single
    Dim InDrIM()  As Integer
    Dim Isort()   As Integer
    Dim MTop()    As Byte
    Dim ki        As Integer
    Dim kk        As Integer
    Dim maiorvar  As Single
    Dim imaiorvar As Integer
    Dim i         As Integer
    Dim j         As Integer

    'Dim Nred As Integer  ' Limite para iniciar o processo de redução
do intervalo

    'Parâmetros de Entrada
    Nbpop = Worksheets("Parametros").Cells(2, 2).Value ' Tamanho da
população de indivíduos
    NPext = Worksheets("Parametros").Cells(3, 2).Value ' Tamanho
máximo da população externa
    Nbgen = Worksheets("Parametros").Cells(4, 2).Value ' Número de
gerações
    Ntype = Worksheets("Parametros").Cells(5, 2).Value ' (1)
Schaffer2;  (2) = Schaffer3 ; (3) Parabolas:

    ' Inicializa variáveis de acordo com o problema:(1) Valor; (2) =
Massa; (3) Valor e Massa
    If Ntype = 1 Then
        ReDim FxI(Nbpop, 2)
        ReDim F1(2)
        ReDim F2(2)
        ReDim avalia(Nbpop, 2)
        ReDim AVAelit(Nbpop + NPext * 2, 2)
        nfunc = 1
```

```
    Nopt = 1
ElseIf Ntype = 2 Then
    ReDim FxI(Nbpop, 2)
    ReDim F1(2)
    ReDim F2(2)
    ReDim avalia(Nbpop, 2)
    ReDim AVAelit(Nbpop + NPext * 2, 2)
    nfunc = 2
    Nopt = 1
End If

nvar = Nopt

ReDim poptemp(nfunc, Nbpop, ngen)
ReDim shar(Nbpop)
ReDim DrI(Nbpop)
ReDim DrIM(Nbpop)
ReDim InDrIM(Nbpop)
ReDim Isort(Nbpop)
ReDim garank(Nbpop)
ReDim X1(Nopt)
ReDim X2(Nopt)
ReDim MaxF(nfunc)
ReDim MinF(nfunc)
ReDim MaxX(Nopt)
ReDim MinX(Nopt)
' Tamanho dos cromossomos
'Nbit = 12
' Tipo de seleção
Iselec = 2 'Amostragem Determ.
' Tipo de cruzamento
Tcruz = 2 'Cruzamento de 2Pt
' Probabilidade de cruzamento
Pcros = Worksheets("Parametros").Cells(6, 2).Value
' Probabilidade de mutação
PMut = Worksheets("Parametros").Cells(7, 2).Value
' Percentagem de elitismo
PNS = Worksheets("Parametros").Cells(9, 2).Value
' Fator de partilha
FatPar = Worksheets("Parametros").Cells(10, 2).Value

adapt = 0

' Número de genes do indivíduo
ReDim MTop(ngen, Nbpop)

' População inicial gerada aleatoriamente
Call gerapop(Nbpop)

' Número de gerações
nger = 1

' Calcula o número de indivíduos a serem processados - SSGA
G = 1
NSel = Fix(G * Nbpop)

ReDim rank(NSel)
ReDim Vmul(Nbit, 1)
ReDim MIDelit(nfunc, Nbpop + NPext * 2, ngen) As Single
```

```
    ' Limita o valor mínimo de NSel em 2 indivíduos
    If (NSel < 2) Then NSel = 2

    NbMIDelit(2) = 0
    ' Loop para processamento das gerações
    While ((nger <= Nbgen) And (NOVO <= NPext))

        ' Avaliação da função de otimização (FxI)

        If Ntype = 1 Then
            Call FuncaoMonoOb 'Atualiza FxI
        Else
            Call FuncaoMultiOb 'Atualiza FxI
        End If

        ' Elitismo
        Call elitismo(NSel, NPext / 2)
        If (ja = 1) Then
            ' Avaliação da função de otimização
            If Ntype = 1 Then
                Call FuncaoMonoOb
            Else
                Call FuncaoMultiOb
            End If
        End If

        ' Aplica o NSGA com as duas funções objetivo básicas
        Call NSGA(NSel, 0)

        ' Para os métodos de seleção e cruzamento
        For i = 1 To ngen
          For j = 1 To Nbpop
            MTop(i, j) = MPop(i, j)      ' População total de
indivíduos
          Next j
        Next i

        ' Calcula o desempenho total
        Dt = 0
        For i = 1 To Nbpop
          Dt = Dt + shar(i)
        Next i

        ' Calcula o desempenho relativo de cada indivíduo - FITNESS
FUNCTION
        For i = 1 To Nbpop
          DrI(i) = shar(i) / Dt
        Next i

        ' Seleção pelo método da Amostragem Determinística
        If (Iselec = 2) Then
            maiorvar = -1E-34
            For i = 1 To Nbpop
              Isort(i) = 0          ' Limpa vetor de indivíduos
              For j = 1 To ngen
                MPop(j, i) = 0              ' Apaga a população a ser
processada
              Next j
```

```
              DrIM(i) = DrI(i) * NSel
              InDrIM(i) = Fix(DrIM(i)) 'número de cópias do individuo
              If InDrIM(i) > maiorvar Then
                maiorvar = InDrIM(i)
                imaiorvar = i
              End If
            Next i
            ki = 1

            ' Seleciona somente o percentual G de indivíduos
            For F = 1 To Nbpop
                kk = InDrIM(F)
                While (kk >= 1) And (ki <= NSel)
                    Isort(ki) = F
                    For i = 1 To ngen
                      MPop(i, ki) = MTop(i, Isort(ki))
                    Next i
                    kk = kk - 1
                    ki = ki + 1
                Wend
            Next F

            For i = 1 To Nbpop
              InDrIM(i) = DrIM(i) - InDrIM(i)
            Next i
            While ki <= NSel
                '[maiorvar,mind] = Max(InDrIM)
                Isort(ki) = imaiorvar
                For i = 1 To ngen
                  MPop(i, ki) = MTop(i, Isort(ki))
                Next i
                InDrIM(imaiorvar) = 0
                ki = ki + 1
            Wend
        End If

        ' Cria nova geração
        nger = nger + 1

        ' Operador cruzamento na população
        Call cruzam(Nbpop, Pcros, Nopt, Tcruz)

        ' Operador mutação na população
        Call mutac(Nbpop, PMut)
    Wend

    Call noequal2(Nbpop)

    Call domina3(Nbpop)

    Call EscreveResultados(Ntype)

End Sub

Sub gerapop(Tpop)
    ' GERAPOP - Gerador de população de indivíduos para Algoritmo
    '       Genético Simples - SGA
    '
    '       Mpop = GERAPOP(Tpop,Ngen)
```

```
    '
    '       Argumentos de Entrada:
    '       Tpop - Tamanho da polpulação de indivíduos.
    '       Ngen - Número de genes por indivíduo.
    '
    '       Argumentos de Saída
    '       Mpop - Matriz mxn com população de indivíduos gerados,
sendo
    '              n indivíduos dispostos em colunas com m genes
dispostos
    '          em linhas.
    ' Copyright (c) Marcos Fonseca - 18/09/99

    ' Gera uma população de Tpop indivíduos com Ngen genes cada
    Dim i As Integer
    Dim j As Integer

    ReDim MPop(ngen, Tpop)
    Randomize (Worksheets("Parametros").Cells(8, 2).Value) 'Le semente
    For i = 1 To ngen
      For j = 1 To Tpop
        MPop(i, j) = Round(Rnd)
      Next j
    Next i
End Sub

Sub FuncaoMonoOb()
  Dim i As Integer
  'Linhas são os individuos na FxI
  MaxF(1) = -1E-34
  MinF(1) = 1E+34
  For i = 1 To Nbpop
      Call Avalia_Individuo(i, False)
      If MaxF(1) < FxI(i, 1) Then MaxF(1) = FxI(i, 1)
      If MinF(1) > FxI(i, 1) Then MinF(1) = FxI(i, 1)
  Next i
End Sub

Sub FuncaoMultiOb()
  Dim i As Integer

  MaxF(1) = -1E-34
  MinF(1) = 1E+34
  MaxF(2) = -1E-34
  MinF(2) = 1E+34

  For i = 1 To Nbpop
    Call Avalia_Individuo(i, True)
    If MaxF(1) < FxI(i, 1) Then MaxF(1) = FxI(i, 1)
    If MaxF(2) < FxI(i, 2) Then MaxF(2) = FxI(i, 2)
    If MinF(1) > FxI(i, 1) Then MinF(1) = FxI(i, 1)
    If MinF(2) > FxI(i, 2) Then MinF(2) = FxI(i, 2)
  Next i
End Sub

Sub Avalia_Individuo(indiv As Integer, Multi As Boolean)
  Dim i As Integer

  For i = 1 To ngen
```

```
    'Marca blocos que necessitam ser extraídos do modelo de blocos
(estéril+minério)
    If MPop(i, indiv) Then
      Call Extract_Block(Int(MID(BlockModel.CodMin(i), 1, 3)),
Int(MID(BlockModel.CodMin(i), 4, 3)), _
                    Int(MID(BlockModel.CodMin(i), 7, 3)))
    End If
  Next i
  Call Pit_Value(indiv, Multi)

End Sub

Sub Pit_Value(indiv As Integer, Multi As Boolean)
  'Atualiza o valor da cava de acordo com os blocos extraídos
  Dim i As Integer
  Dim j As Integer
  Dim k As Integer
  BlockModel.Value = 0
  BlockModel.Mass = 0
  For i = 1 To BlockModel.NRow
    For j = 1 To BlockModel.NCol
      For k = 1 To BlockModel.NLevel
        If BlockModel.Extract(i, j, k) Then
          BlockModel.Value = BlockModel.Value + BlockModel.Values(i,
j, k)
          If BlockModel.Values(i, j, k) >= 0 Then BlockModel.Mass =
BlockModel.Mass + 1
          BlockModel.Extract(i, j, k) = False
        End If
      Next k
    Next j
  Next i
  If Multi Then
    FxI(indiv, 1) = BlockModel.Value
    FxI(indiv, 2) = BlockModel.Mass
  Else
    FxI(indiv, 1) = Pond(1) * BlockModel.Value + Pond(2) *
BlockModel.Mass
  End If

End Sub

Sub Extract_Block(i As Integer, j As Integer, k As Integer)
  'Lavra os blocos de acordo com índices  ijk e do angulo geral de
talude
  Dim L As Integer
  Dim N As Integer

  Call Extract_Block_Above(i, j, k)

  N = 1
  For L = k - 1 To 1 Step -1
    Call Extract_Block_Around(i, j, L, N)
    N = N + 1
  Next L
End Sub

Sub Extract_Block_Around(i As Integer, j As Integer, k As Integer, N
As Integer)
```

```
  'Lavra os blocos de acordo com o ângulo de talude
  Dim TanAngle As Single
  Dim Pi       As Single
  Dim iRow     As Integer
  Dim iRowMax  As Integer
  Dim iRowMin  As Integer
  Dim iCol     As Integer
  Dim iColMax  As Integer
  Dim iColMin  As Integer
  Dim Aux      As Single

  Pi = 3.14159265358979
  TanAngle = Tan(BlockModel.Angle * Pi / 180)

  iRowMax = i + (BlockModel.SizeBZ / BlockModel.SizeBY + 1) * N
  If iRowMax > BlockModel.NRow Then iRowMax = BlockModel.NRow
  iRowMin = i - (BlockModel.SizeBZ / BlockModel.SizeBY + 1) * N
  If iRowMin < 1 Then iRowMin = 1
  iColMax = j + (BlockModel.SizeBZ / BlockModel.SizeBX + 1) * N
  If iColMax > BlockModel.NCol Then iColMax = BlockModel.NCol
  iColMin = j - (BlockModel.SizeBZ / BlockModel.SizeBX + 1) * N
  If iColMin < 1 Then iColMin = 1

  For iRow = iRowMin To iRowMax
    For iCol = iColMin To iColMax
      Aux = Sqr((i - iRow) * BlockModel.SizeBY * (i - iRow) *
BlockModel.SizeBY + (j - iCol) * BlockModel.SizeBX * (j - iCol) *
BlockModel.SizeBX)
      If (Aux <= (BlockModel.SizeBZ * N / TanAngle)) Then
        BlockModel.Extract(iRow, iCol, k) = True
      End If
    Next iCol
  Next iRow
End Sub

Sub Extract_Block_Above(i As Integer, j As Integer, k As Integer)
  'Lavra os blocos acima do bloco atual
  Dim L As Integer
  If (i <= BlockModel.NRow) And (j <= BlockModel.NCol) And (k <=
BlockModel.NLevel) And (i > 0) And (j > 0) Then
    For L = 1 To k
      BlockModel.Extract(i, j, L) = True
    Next L
  End If
End Sub

Sub elitismo(NSel, NPext)
    Dim i        As Integer
    Dim j        As Integer
    Dim z        As Integer
    Dim NS       As Integer
    Dim r()      As Integer
    Dim T()      As Integer
    Dim NOVO     As Integer
    Dim ind      As Integer
    Dim Verifica As Integer
    ReDim r(NSel)
    ReDim T(NSel)
```

```
    ' Verifica se o tamanho da população externa não excedeu Pext
(tamanho máximo)
    NOVO = 0
    ja = 0
    If (NbMIDelit(2) <= NPext) Then
        ' acrescenta os indivíduos eficientes à população externa
        Call domina1(NSel)
        For j = NbMIDelit(1) To NbMIDelit(2)
            For i = 1 To nvar
              For z = 1 To ngen
                MIDelit(i, j, z) = poptemp(i, j - NbMIDelit(1) + 1, z)
              Next z
            Next i
        Next j
        For i = NbMIDelit(1) To NbMIDelit(2)
            For j = 1 To nfunc
                AVAelit(i, j) = avalia(i - NbMIDelit(1) + 1, j)
            Next j
        Next i
    Else
        'acrescenta novos indivíduos eficientes à Pext e elimina os
ineficientes
        Call domina1(NSel)
        For j = NbMIDelit(1) To NbMIDelit(2)
            For i = 1 To nvar
                For z = 1 To ngen
                  MIDelit(i, j, z) = poptemp(i, j - NbMIDelit(1) + 1,
z)
                Next z
            Next i
        Next j

        For i = NbMIDelit(1) To NbMIDelit(2)
            For j = 1 To nfunc
                AVAelit(i, j) = avalia(i - NbMIDelit(1) + 1, j)
            Next j
        Next i
        Call domina2(NbMIDelit(2))
        'retira individuos iguais
        Call noequal1
        NOVO = NbMIDelit(2)
        ' realiza substituição de PNS (porcento) de elementos de MID
por MIDelit
        NS = Round(NSel * PNS)

        ' limita o # de indiv substituídos pelo tamanho da população
externa
        If (NS > NOVO) Then
            NS = NOVO
        End If

        ' inicializa variáveis
        For i = 1 To NS
          T(i) = 0
          r(i) = 0
        Next i
        ind = 1
        ReDim T(NbMIDelit(2))
        ' substitui o # de indivíduos desejados
```

```
        While (ind <= NS)
            r(ind) = Int((Rnd * NSel) + 1)
            T(ind) = Int(Rnd * NbMIDelit(2) + 1)
            Verifica = 1
            ' verifica se indivíduo já foi substituido
            For i = 1 To (ind - 1)
                If ((r(i) = r(ind)) And (T(i) = T(ind))) Then
                    Verifica = 0
                End If
            Next i
            ' substitui inidivíduo
            If Verifica = 1 Then
                For i = 1 To nvar
                    For z = 1 To ngen
                        'Xb10(i, r(ind)) = MIDelit(i, T(ind))
                        MPop(z, r(ind)) = MIDelit(i, T(ind), z)
                    Next z
                Next i
                ind = ind + 1
                ja = 1
            End If
        Wend
    End If
End Sub

Sub domina1(nindiv) 'atualiza matrizes poptemp e avalia (argumentos
FxI e Xb10)
  Dim i              As Integer
  Dim j              As Integer
  Dim k              As Integer
  Dim c_rank         As Integer
  Dim c_dom          As Integer
  Dim rank()         As Integer
  ReDim rank(nindiv) As Integer
  Dim c              As Integer
  Dim z              As Integer

    ' Cobre toda a população corrente
    For i = 1 To nindiv
        ' Classificação para o indivíduo i
        c_rank = 0
        'Compara indivíduo "i" com a população usando a definição de
dominância
        For j = 1 To nindiv
            If i <> j Then
                ' Define dominância
                c_dom = 1
                ' Conceito de dominancia e empregado aqui.
                ' Dimensão de FxI(a,b   a: Tamanho da População
                '                       b: Número de funções objetivo
                ' Exemplo: FxI(1,2):    Valor da função objetivo 2 do
indivíduo 1.
                '
                For k = 1 To nfunc
                    If (FxI(i, k) > FxI(j, k)) Then c_dom = c_dom * 0
                    If (FxI(i, k) = FxI(j, k)) Then c_dom = c_dom * 1
                    If (FxI(i, k) < FxI(j, k)) Then c_dom = c_dom * 2
                Next k 'Fim do conceito de dominância
                'Checa "i" é eficiente
```

```
                If (c_dom > 1) Then c_rank = c_rank + 1
            End If 'Fim da comparação do indivíduo atual com a
população
        Next j
        ' Classificação do indivíduo "i" pelo conceito de dominância
        rank(i) = 1 + c_rank
    Next i
    c = 1
    For i = 1 To nindiv
        If (rank(i) = 1) Then
            For j = 1 To nvar
              For z = 1 To ngen
                'poptemp(j, c) = Xb10(j, i) sapos
                poptemp(j, c, z) = MPop(z, i)
              Next z
            Next j
            For j = 1 To nfunc
              avalia(c, j) = FxI(i, j)
            Next j
            c = c + 1
        End If
    Next i
    If c > 1 Then
      NbMIDelit(1) = NbMIDelit(2) + 1
      NbMIDelit(2) = NbMIDelit(2) + c - 1
    End If
End Sub

Sub domina2(nindiv) 'atualiza matrizes poptemp e avalia (argumentos
AVAelit e MIDElit)
  Dim i              As Integer
  Dim j              As Integer
  Dim k              As Integer
  Dim z              As Integer
  Dim c_rank         As Integer
  Dim c_dom          As Integer
  Dim rank()         As Integer
  ReDim rank(nindiv) As Integer
  Dim c              As Integer

    ' Cobre toda a população corrente
    For i = 1 To nindiv
        ' Classificação para o indivíduo i
        c_rank = 0
        'Compara indivíduo "i" com a população usando a definição de
dominância
        For j = 1 To nindiv
            If i <> j Then
                ' Define dominância
                c_dom = 1
                ' Conceito de dominancia e empregado aqui.
                ' Dimensão de FxI(a,b   a: Tamanho da População
                '                       b: Número de funções objetivo
                ' Exemplo: FxI(1,2):    Valor da função objetivo 2 do
indivíduo 1.
                '
                For k = 1 To nfunc
                    If (AVAelit(i, k) > AVAelit(j, k)) Then c_dom =
c_dom * 0
```

```
                    If (AVAelit(i, k) = AVAelit(j, k)) Then c_dom =
c_dom * 1
                    If (AVAelit(i, k) < AVAelit(j, k)) Then c_dom =
c_dom * 2
                Next k 'Fim do conceito de dominância
                'Checa "i" é eficiente
                If (c_dom > 1) Then c_rank = c_rank + 1
            End If 'Fim da comparação do indivíduo atual com a
população
        Next j
        ' Classificação do indivíduo "i" pelo conceito de dominância
        rank(i) = 1 + c_rank
    Next i
    c = 1
    For i = 1 To nindiv
        If (rank(i) = 1) Then
            For j = 1 To nvar
                For z = 1 To ngen
                  MIDelit(j, c, z) = MIDelit(j, i, z)
                Next z
            Next j
            For j = 1 To nfunc
              AVAelit(c, j) = AVAelit(i, j)
            Next j
            c = c + 1
        End If
    Next i
    If c > 1 Then
      NbMIDelit(1) = 1
      NbMIDelit(2) = c - 1
    End If
End Sub

Sub domina3(nindiv) 'atualiza matrizes poptemp e avalia (argumentos
FxI e Xb10)
  Dim i              As Integer
  Dim j              As Integer
  Dim k              As Integer
  Dim z              As Integer
  Dim c_rank         As Integer
  Dim c_dom          As Integer
  Dim rank()         As Integer
  ReDim rank(nindiv) As Integer
  Dim c              As Integer

    ' Cobre toda a população corrente
    For i = 1 To nindiv
        ' Classificação para o indivíduo i
        c_rank = 0
        'Compara indivíduo "i" com a população usando a definição de
dominância
        For j = 1 To nindiv
            If i <> j Then
                ' Define dominância
                c_dom = 1
                ' Conceito de dominancia e empregado aqui.
                ' Dimensão de FxI(a,b   a: Tamanho da População
                '                       b: Número de funções objetivo
```

```
                ' Exemplo: FxI(1,2):    Valor da função objetivo 2 do
indivíduo 1.
                '
                For k = 1 To nfunc
                    If (FxI(i, k) > FxI(j, k)) Then c_dom = c_dom * 0
                    If (FxI(i, k) = FxI(j, k)) Then c_dom = c_dom * 1
                    If (FxI(i, k) < FxI(j, k)) Then c_dom = c_dom * 2
                Next k 'Fim do conceito de dominância
                'Checa "i" é eficiente
                If (c_dom > 1) Then c_rank = c_rank + 1
            End If 'Fim da comparação do indivíduo atual com a
população
        Next j
        ' Classificação do indivíduo "i" pelo conceito de dominância
        rank(i) = 1 + c_rank
    Next i
    c = 1
    For i = 1 To nindiv
        If (rank(i) = 1) Then
            For j = 1 To nvar
              For z = 1 To ngen
                'Xb10(j, c) = Xb10(j, i)
                MPop(z, c) = MPop(z, i)
              Next z
            Next j
            For j = 1 To nfunc
                FxI(c, j) = FxI(i, j)
            Next j
            c = c + 1
        End If
    Next i
End Sub

Sub NSGA(nindiv, NICHOS)
    '
    ' O algoritmo NSGA é baseado em um procedimento de classificação
dos pontos eficientes da população
    '
    ' Antes do Operador Seleção Atuar, são realizados os seguintes
passos:
    '
    '   (1)  População é classificada com base no conceito de
dominância. Todos os indivíduos inicialmente
    '        classificados assumem um valor de função de aptidão
inicial (DUMMY FITNESS). Para manter a
    '        diversidade, estes indivíduos são submetidos a uma função
de partilha triangular.
    '
    '   (2)  Os indivíduos anteriormente classificados são
"esquecidos" da população e o processo de
    '        classificação é iniciado novamente. A função de aptidão
desta segunda classificação deve
    '        ser menor que o MENOR valor de aptidão da classificação
anterior
    '
    '   (3)  O processo continua até todos os indivíduos serem
classificados
    '
```

```
    '   Este processo de classificação dará origem à fronteiras
eficientes na
    '   população corrente.
    '
    '   Argumentos de Entrada do NSGA:
    '       F -  Matriz com os valores de cada função objetivo para
toda a população
    ' nindiv -  Número de indivíduos da população a serem analisados
    '   Lvar -  Vetor com os limites das variáveis de otimização
    '   nvar -  Número de variáveis de otimização do problema
    '  nfunc -  Número de funções objetivo a serem usadas na definição
de eficiência
    '    MID -  Matriz com dados das variáveis de otimização
decodificadas
    ' NICHOS -  Tipo da técnica de nichos a ser utilizada pelo NSGA
    '   Argumentos de saída:
    '   MFxI -  Vetor com a nova função de aptidão da população
    ' Copyright (c) Alexandre Henrique Farah Dias - 15/07/00

    Dim i           As Integer
    Dim j           As Integer
    Dim k           As Integer
    Dim Ssort()     As Single
    Dim c_Sort     As Integer
    Dim dshared     As Single
    Dim ini_dummy_f As Single
    Dim del_dummy_f As Single
    Dim min_dummy_f As Single
    Dim c_rank      As Integer
    Dim nfront      As Integer
    Dim fron()      As Integer
    Dim pop_ndom()  As Integer
    Dim c_dom       As Integer
    ReDim Ssort(nindiv)
    ReDim fron(nindiv)
    ReDim pop_ndom(nindiv)

    ' Inicialização das variáveis
    For i = 1 To nindiv
      garank(i) = 0
      shar(i) = 0
      Ssort(i) = 0
    Next i

    ' Fronteira Pareto Ótimo classificada
    nfront = 0

    ' Valores iniciais para função de partilha triangular
    ini_dummy_f = 100
    del_dummy_f = 0.05 * ini_dummy_f

    ' Raio de partilha estimado
    If FatPar = 0 Then
      dshared = 0.5 / Application.WorksheetFunction.Power(nindiv, 1 /
nvar)
    Else
      dshared = FatPar
    End If
```

```
    c_Sort = 0

    ' Realiza definição das fronteiras PO
    While (c_Sort < nindiv)
        ' Cobre toda a população corrente
        For i = 1 To nindiv
            ' Classificação para o indivíduo i
            c_rank = 0
            ' Indivíduo i ainda não rankeado
            If (Ssort(i) = 0) Then
                'Compara indivíduo "i" com a população usando a
definição de dominância
                For j = 1 To nindiv
                    'Indivíduo j ainda não rankeado
                    If ((Ssort(j) = 0) And (i <> j)) Then
                        ' Define dominância
                        c_dom = 1
                        ' Conceito de dominancia e empregado aqui.
                        For k = 1 To nfunc
                            If (FxI(i, k) > FxI(j, k)) Then c_dom =
c_dom * 0
                            If (FxI(i, k) = FxI(j, k)) Then c_dom =
c_dom * 1
                            If (FxI(i, k) < FxI(j, k)) Then c_dom =
c_dom * 2
                        Next k
                    End If 'Fim do conceito de dominância
                    ' Checa se "j" domina "i"
                    If (c_dom > 1) Then c_rank = c_rank + 1
                Next j
            End If 'Fim da comparação do indivíduo atual com a
população
            ' Classificação do indivíduo "i" pelo conceito de
dominância
            garank(i) = 1 + c_rank
        Next i

        ' Atualiza número de fronteiras
        nfront = nfront + 1
        k = 0

        ' Determina valor da avaliação para a fronteira corrente
        For i = 1 To nindiv
            ' Indivíduo "i" é eficiente?
            If (garank(i) = 1) Then
                'Total de indivíduos ordenados até o momento
                c_Sort = c_Sort + 1
                'Total de indivíduos da fronteira atual
                k = k + 1
                Ssort(i) = 1       'Indica que indivíduo já foi
classificado
                garank(i) = -2      'Desabilita flag de classificacao
                fron(i) = nfront  'Indica fronteira do indivíduo
classificado
                ' Função de aptidão do indivíduo
                If (nfront = 1) Then
                    shar(i) = ini_dummy_f 'Se é a primeira fronteira,
assume um valor default
                Else
```

```
                    shar(i) = min_dummy_f - del_dummy_f
                End If
            End If
        Next i

        ' Determina valor da avaliação para a fronteira corrente
        pop_ndom(nfront) = k

        ' Realiza a técnica de formaçào de nichos para os indivíduos
classificados
        Call nichoAG(NICHOS, nindiv, dshared)

        ' Atualiza classificação corrente
        For i = 1 To nindiv
          If (garank(i) = -2) Then
             garank(i) = -1
          End If
        Next i
    Wend
End Sub

Sub cruzam(Nbpop, Pcru, Nopt, Tcruz)

    ' CRUZAM  - Cruzamento de indivíduos de uma população para
    '       Algoritmo Genético Simples. Não é permitido o cruzamento
    '       entre os mesmos indivíduos. O ponto de corte é escolhido
    '       aleatoriamente. São permitidos múltiplos cruzamentos de
    '       um mesmo casal.
    '
    '       MIC = CRUZAM(Mpop,Pcru)
    '
    '       Argumentos de Entrada:
    '       Mpop - Matriz mxn com a população de n indivíduos
    '                  dipostos em colunas.
    '       Pcru - Probabilidade de cruzamento para cada indivíduo,
    '                  com valor entre 0 e 1.
    '
    '       Argumentos de Saída
    '       MIC -  Matriz mxn com a nova população resultante do
cruzamento,
    '          sendo n indivíduos dipostos em colunas.
    ' Copyright (c) Marcos Fonseca - 19/09/99

    Dim F  As Integer
    Dim ii As Integer
    Dim v  As Integer
    Dim Vind() As Integer
    Dim Pind() As Single ' Probabilidade de cada indivíduo
    Dim Icru() As Single ' Indivíduos que cruzam
    Dim i As Integer
    Dim j As Integer
    Dim Npop() As Byte
    Dim Pcor1 As Integer
    Dim Pcor2 As Integer
    Dim Cut1 As Integer
    Dim Cut2 As Integer
    Dim Pai As Integer
    Dim Mae As Integer
    Dim Aux As Integer
```

```
    Dim Ncru As Integer
    Dim vpar() As Integer ' Vetor de parceiros

    ReDim Vind(Nbpop)
    ReDim Pind(Nbpop)
    ReDim Icru(Nbpop)
    ReDim Npop(ngen, Nbpop)
    ReDim vpar(Nbpop)

    ' Vetor de indivíduos
    j = 0
    For i = 1 To Nbpop
      Vind(i) = i
      Pind(i) = Int((Nbpop) * Rnd + 1)
      If Pind(i) <= Pcru Then
        j = j + 1
        Icru(j) = i
      End If
    Next i

    ' Vetor com os indivíduos que cruzam
    Ncru = j        ' Número de cruzamentos
    For i = 1 To ngen
      For j = 1 To Nbpop
        Npop(i, j) = MPop(i, j)  ' Nova população
      Next j
    Next i

    If Ncru > 0 Then              ' Continua somente se existirem
cruzamentos
    ' Parceiros para formação do casal
        For F = 1 To Ncru
             vpar(F) = Int((Nbpop) * Rnd + 1)
        Next F
        If Tcruz = 2 Then
            ' Cruzamento com 2 Pontos de corte para cada indivíduo que
cruza
            Pcor1 = Round(Int((Ncru) * Rnd + 1) * (ngen - 2)) + 1
            Pcor2 = Pcor1
            While Pcor2 = Pcor1
                Pcor2 = Round(Int((Ncru) * Rnd + 1) * (ngen - 2)) + 1
            Wend
            ' Faz o cruzamento
            For F = 1 To Ncru
                Pai = Icru(F)
                Mae = vpar(F)
                Cut1 = Pcor1
                Cut2 = Pcor2
                If Cut1 > Cut2 Then
                    Aux = Cut1
                    Cut1 = Cut2
                    Cut2 = Aux
                End If
                If Cut1 = Cut2 Then
                    If Cut1 = ngen Then
                        Cut1 = Cut1 - 1
                    ElseIf Cut2 = 1 Then
                        Cut2 = Cut2 + 1
                    End If
```

```
                End If
                For i = Cut1 To Cut2
                    Npop(i, Pai) = MPop(i, Mae)
                    Npop(i, Mae) = MPop(i, Pai)
                Next i
            Next F
        End If
    End If
    For i = 1 To ngen
      For j = 1 To Nbpop
        MPop(i, j) = Npop(i, j)  ' Atualiza Mpop
      Next j
    Next i
End Sub

Sub mutac(Nbpop, PMut)
    ' MUTAC   - Mutação dos bits (genes) de indivíduos de uma
população para
    '       Algoritmo Genético Simples.
    '
    '       Mpop - Matriz mxn com a população de n indivíduos
    '                   dipostos em colunas.
    '       Pmut - Probabilidade de mutação para cada bit (gene) dos
indivíduos,
    '                   com valor entre 0 e 1.

    Dim i As Integer
    Dim j As Integer

        ' Mutação bit a bit com probabilidade 100'/No. de bits do
individuo
        ' Indivíduos e bits que sofrerão mutação
        For i = 1 To ngen
          For j = 1 To Nbpop
            If Rnd <= PMut Then
                MPop(i, j) = MPop(i, j) * (-1) + 1
            End If
          Next j
        Next i
End Sub

Sub noequal1()
  Dim i    As Integer
  Dim j    As Integer
  Dim k    As Integer
  Dim z    As Integer
  Dim N    As Integer
  Dim Acum As Single
    i = 1
    While i <= NbMIDelit(2)
        For j = NbMIDelit(2) To i + 1 Step -1
            Acum = 0
            For k = 1 To nfunc
              Acum = Acum + (AVAelit(i, k) - AVAelit(j, k)) ^ 2
            Next k
            If Sqr(Acum) < 0.1 Then
                For N = j To NbMIDelit(2)
                    For k = 1 To nfunc
                        AVAelit(N, k) = AVAelit(N + 1, k)
```

```
                    Next k
                    For k = 1 To nvar
                        For z = 1 To ngen
                          MIDelit(k, N, z) = MIDelit(k, N + 1, z)
                        Next z
                    Next k
                Next N
                NbMIDelit(2) = NbMIDelit(2) - 1
            End If
        Next j
        i = i + 1
    Wend
End Sub

Sub noequal2(Nbpop)
  Dim i    As Integer
  Dim j    As Integer
  Dim k    As Integer
  Dim z   As Integer
  Dim N    As Integer
  Dim Acum As Single
    i = 1
    While i <= Nbpop
        For j = Nbpop To i + 1 Step -1
            Acum = 0
            For k = 1 To nfunc
              Acum = Acum + (FxI(i, k) - FxI(j, k)) ^ 2
            Next k
            If Sqr(Acum) < 0.1 Then
                For N = j To Nbpop
                    For k = 1 To nfunc
                      If N < Nbpop Then FxI(N, k) = FxI(N + 1, k)
'Falta
                    Next k
                    For k = 1 To nvar
                      If N < Nbpop Then
                        'Xb10(k, N) = Xb10(k, N + 1)
                        For z = 1 To ngen
                          MPop(z, N) = MPop(z, N + 1)
                        Next z
                      End If
                    Next k
                Next N
                Nbpop = Nbpop - 1
            End If
        Next j
        i = i + 1
    Wend
End Sub

Sub nichoAG(TIPOPAR, nindiv, dshared)
    ' Esta função realiza a função de partilha triangular a partir dos
    ' valores assumidos durante a classificacao da fronteira corrente
    '
    ' Argumentos de Entrada da nichosAG:
    '
    ' TIPOPAR - Tipo da técnica de nicho a ser empregada
    '     MID - Matriz com variáveis de otimização decodificadas
    '       F - Matriz com dados das funções objetivo
```

```
    '    Lvar - Vetor com os limites das variáveis de otimização
    '    shar - Vetor classificação degradada da população em
fronteiras PO
    '    rank - Vetor de classificação da população em fronteiras PO
    '  nindiv - Número de indivíduos na população
    '    nvar - Número de variáveis
    '   nfunc - Número de funções a serem usadas na definição de
eficiência
    ' dshared - Valor máximo do raio de partilha para indivíduo
pertencer ao nicho
    ' Argumentos de Saída da nichosAG:
    '    shar - Vetor com a nova função de partilha da fronteira PO
atual
    ' Copyright (c) Alexandre Henrique Farah Dias - 15/07/00

    ' Define os limites para aplicação da técnica de nicho
    Dim i           As Integer
    Dim j           As Integer
    Dim k           As Integer
    Dim nichecount  As Integer
    Dim d           As Single

    For i = 1 To nvar
      MaxX(i) = Lvar(2, i)
      MinX(i) = Lvar(1, i)
    Next i

    d = 0

    ' Aplicação da técnica de formação de nichos
    For i = 1 To nindiv
        ' Número de indivíduos pertencentes ao nicho
        nichecount = 1
        ' Fronteira Pareto Ótimo atual
        If (garank(i) = -2) Then
            For j = 1 To nindiv
                If (i <> j) Then
                    ' Indivíduo pertencente a PO atual?
                    If (garank(j) = -2) Then
                        ' Variáveis da dimensão
                        For k = 1 To nfunc
                            F1(k) = FxI(i, k)
                            F2(k) = FxI(j, k)
                        Next k
                        For k = 1 To nvar
                            'X1(k) = Xb10(k, i) sapos
                            'X2(k) = Xb10(k, j)
                        Next k
                        ' Determina distância entre os dois indivíduos
cujo espaço é dado pela variável TIPOPAR
                        d = distanciaAG(1)
                        ' Aplica-se a técnica de formação de nichos
                        If (d < dshared) Then
                            nichecount = nichecount + (1 - (d /
dshared) * (d / dshared))
                        End If
                    End If
                End If
            Next j
```

```
        End If
        ' Recalcula valor de aptidão usando a informação dos
indivíduos
        ' pertencentes ao nicho.
        shar(i) = (shar(i) / nichecount)
    Next i
End Sub

Function distanciaAG(TIPOPAR) As Single
    ' Esta função calcula a distância entre dois indivíduos a partir
    ' do parâmetro TIPOPAR que seleciona o espaço para medição
    '
    ' Argumentos de entrada da distanciaAG:
    ' TIPOPAR - Espaço utilizado no cálculo da distância
    '      F1 - Vetor de funções para indivíduo 1
    '      F2 - Vetor de funções para indivíduo 2
    '      X1 - Vetor das variáveis de otimização para indivíduo 1
    '      X2 - Vetor das variáveis de otimização para indivíduo 2
    '    MaxF - Valores máximos das funções objetivo na geração
corrente
    '    MinF - Valores mínimos das funções objetivo na geração
corrente
    '    MaxX - Valores máximos das variáveis
    '    MinX - Valores mínimos das variáveis
    '
    ' Argumentos de saída da distanciaAG:
    '       d - Distância calculada
    ' Copyright (c) Alexandre Henrique Farah Dias - 15/07/00

    Dim k   As Integer
    Dim dst As Single

    dst = 0

    ' calcula a distância entre dois indivíduos
    If TIPOPAR = 0 Then ' espaço das variáveis de otimização
        ' Partilha no espaço de parâmetros
        For k = 1 To nvar
            dst = dst + (X1(k) - X2(k)) ^ 2 / ((MaxX(k) - MinX(k)) ^
2)
        Next k
    Else  ' espaço das funções objetivo
        ' Partilha no espaço de funções
        For k = 1 To nfunc
            If (MaxF(k) - MinF(k)) ^ 2 > 0 Then
              dst = dst + (F1(k) - F2(k)) ^ 2 / ((MaxF(k) - MinF(k)) ^
2)
            Else
              dst = 0
            End If
        Next k
    End If

    ' Distância medida
    dst = Application.WorksheetFunction.Power(dst, 0.5)
    distanciaAG = dst
End Function

Sub EscreveResultados(Ntype As Integer)
```

```
    ' Atualiza pasta de "Resultados" com as soluções encontradas pelo
AG
    Dim i As Integer
    Dim j As Integer
    Dim k As Integer
    Dim Coluna As Integer

    If Ntype = 1 Then
      Coluna = 0
    ElseIf Ntype = 2 Then
      Coluna = 3
    Else
      Coluna = 6
    End If
    For i = 1 To Nbpop
      For j = 1 To nfunc
        For k = 1 To nvar
          Worksheets("Resultados").Cells(i + 2, 1).Value = i
          If Ntype = 1 Then
            If i = 1 Then
              Worksheets("Resultados").Cells(i + 2, j + 1).Value =
FxI(i, j)
              Call Atualiza_Desenho_Cava(i, 0)
            End If
          ElseIf Ntype = 2 Then
            'Worksheets("Resultados").Cells(i + 2, k + 4).Value =
Xb10(k, i)
            Worksheets("Resultados").Cells(i + 2, j + 5).Value =
FxI(i, j)
            Call Atualiza_Desenho_Cava(i, 3)
          End If
        Next k
      Next j
    Next i
End Sub
Sub Clear_Sheet(Name As String)
  'Apaga a planilha repassada em Name
  Dim NewSheet As Worksheet
  Set NewSheet = Worksheets.Add
  NewSheet.Name = Name

  With Worksheets(Name)
    Sheets(Name).Select
    Sheets(Name).Activate
    Cells.Select
    Selection.ClearContents
    Range("A1").Select
    ActiveWindow.DisplayGridlines = False
  End With
End Sub

Sub Atualiza_Desenho_Cava(indiv As Integer, Coluna As Integer)
  'Calcula Valores de Valor, Massa e desenha cava final
  Dim i As Integer
  Dim j As Integer
  Dim k As Integer


  For i = 1 To BlockModel.NRow
```

```
    For j = 1 To BlockModel.NCol
      For k = 1 To BlockModel.NLevel
        BlockModel.Extract(i, j, k) = False
      Next k
    Next j
  Next i

  For i = 1 To ngen
    'Marca blocos que necessitam ser extraídos do modelo de blocos
(estéril+minério)
    If MPop(i, indiv) Then
      Call Extract_Block(Int(MID(BlockModel.CodMin(i), 1, 3)),
Int(MID(BlockModel.CodMin(i), 4, 3)), _
                  Int(MID(BlockModel.CodMin(i), 7, 3)))
    End If
  Next i
  BlockModel.Value = 0
  BlockModel.Mass = 0
  For i = 1 To BlockModel.NRow
    For j = 1 To BlockModel.NCol
      For k = 1 To BlockModel.NLevel
        If BlockModel.Extract(i, j, k) Then
          BlockModel.Value = BlockModel.Value + BlockModel.Values(i,
j, k)
          If BlockModel.Values(i, j, k) >= 0 Then BlockModel.Mass =
BlockModel.Mass + 1
          If Coluna = 0 Then Worksheets("Resultados").Cells(k, 11 +
j).Value = True
        Else
          If Coluna = 0 Then Worksheets("Resultados").Cells(k, 11 +
j).Value = False
        End If
      Next k
    Next j
  Next i
  Worksheets("Resultados").Cells(indiv + 2, 3 + Coluna).Value =
BlockModel.Value
  Worksheets("Resultados").Cells(indiv + 2, 4 + Coluna).Value =
BlockModel.Mass
End Sub
```