UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA DE SISTEMAS

BUSCA TABU NA SOLUÇÃO DE PROBLEMAS DE PROGRAMAÇÃO ZERO-UM

FELIPE MARTINS MULLER

Este exemplar corresponde à redação final da tese defendida por Felipe Martins Muller e aprovada pela Comissão Julgadora em 11/04/90.

Orientador:

Prof. Dr. Paulo Morelato França

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas - UNICAMP, como parte dos requisitos exigidos para a obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA.

- ABRIL 1990 -

AGRADECIMENTOS

A todos que colaboraram na realização deste trabalho e especialmente

- ao Paulo França, pela orientação durante o desenvolvimento da tese, pela sua ajuda e estímulo a meu trabalho de pesquisa e por sua amizade

- aos professores e funcionários do DENSIS, pela acolhida e amizade

- a Sabrina, pelo amor e dedicação em todos os momentos

- a Pedro e Heloisa, meus pais, pelo constante interesse, preocupação e amor

- ao Celso, pelo companheirismo e auxílio desde o início até o fim desta jornada

- a *gurizada medonha*, pelas horas de diversão e alegria

Esta tese contou com apoio financeiro da CAPES, CNPQ e UFSM.

RESUMO

Este trabalho trata da resolução do problema de programação linear com variáveis zero-um através da adaptação das técnicas de Busca Tabu à uma heurística clássica de Pivot e Complemento. É mostrado em detalhes como é construido o novo método e como é o seu comportamento computacional sobre um conjunto de problemas reais e um conjunto de problemas gerados aleatóriamente. O enfoque apresentado mostra-se promissor para resolver problemas de grande porte, apresentando em todos os casos testados, soluções de qualidade superior em relação à heurística clássica, especialmente para problemas altamente restritivos.

ÍNDICE

INTRODUÇÃO

Este trabalho enfoca um método heurístico desenvolvido para resolver uma classe específica de problemas de programação 0-1.

Matematicamente o problema pode ser escrito como:

$$(P) \qquad \text{Max} \quad cx$$

$$\text{s.a.} \quad Ax \leq b$$

$$x_j \in \{ 0 , 1 \} , \quad j \in N$$

onde: c é um vetor (1xn);

A é uma matriz (mxn);

b é um vetor (mx1);

N = { 1, ... , n }.

(P) é conhecido como problema de programação linear com variáveis bivalentes, onde se supõe que a matriz A não tem nenhuma particularidade estrutural.

Os métodos exatos desenvolvidos para esta classe de problemas, em geral fazem uso de algoritmos enumerativos ou de planos de corte. Os métodos heurísticos, que não garantem a

otimalidade, fazem uso de procedimentos dos mais variados tipos, alguns dos quais serão mostrados no Capítulo 1 deste trabalho.

Uma dificuldade intrínseca aos métodos heurísticos é a sua inabilidade em saber sair dos ótimos locais, que funcionam como verdadeiras armadilhas para estes métodos. Eles trabalham baseados em decisões seqüenciais que proporcionam melhorias na função objetivo. Quando melhorias não são mais possíveis, caracterizando um ótimo local, é preciso abandonar a direção de busca e reinicializar a heurística, o que, em geral não produz bons resultados. Uma alternativa é alargar o espectro de busca, o que a torna mais demorada. Poder-se-ia dizer que o bom desempenho da heurística está relacionado não só a sua habilidade em chegar rápido a ótimos locais, mas também saber sair deles de forma inteligente, a fim de atingir soluções as mais próximas possíveis do ótimo global. Em outras palavras, um bom método heurístico deve ser eficiente e eficaz.

Nesta linha de pesquisa as contribuições de GLOVER [12], [14] e [15] são significativas. Ele propõe a Busca Tabu que pode ser vista como uma meta-heurística, superimposta a outra heurística. Esta estratégia busca transcender a otimalidade local, proibindo, ou mais amplamente, penalizando certos movimentos que levariam a regiões de busca já pesquisadas e com isto evitando ciclagens.

Neste trabalho é aplicada a estratégia da busca tabu sobre a heurística de BALAS e MARTIN [4] para resolver problemas de programação 0-1. A heurística de Balas e Martin é uma heurística tradicional que, apesar de ter boas características gerais de eficiência e eficácia, é aprisionada em ótimos locais pobres quando aplicada em algumas classes especiais de problemas. O objetivo perseguido é dotar a heurística tradicional de um dispositivo mais poderoso capaz de levar, especialmente os problemas de grande porte e/ou altamente restritos, a soluções mais próximas da ótima.

A maneira como isto foi feito, bem como o algoritmo desenvolvido serão apresentados no Capítulo 2 deste trabalho.

No capítulo 3 são apresentadas mudanças no sentido de melhorar o desempenho do algoritmo anterior. Neste sentido foram estudados testes mais eficientes e estruturas de dados especiais adaptáveis ao algoritmo.

Finalizando este trabalho é apresentada uma bateria de testes realizada numa seleção de problemas, na qual é analisado o comportamento do método proposto.

CAPÍTULO 1

REVISÃO BIBLIOGRÁFICA DE PROGRAMAÇÃO 0-1 HEURÍSTICA

Programação 0-1 trata de resolver modelos matemáticos onde todas as variáveis são binárias. Uma variadíssima classe de problemas provenientes de diversas áreas do conhecimento podem ser formulados através de modelos de programação 0-1. Os problemas de natureza combinatorial, geralmente, são formulados com o auxílio deste tipo de variável.

Os métodos chamados exatos, que garantem a convergência para uma solução ótima do modelo, encontram sérias dificuldades para solucionar problemas práticos de programação 0-1. Muitos problemas típicos de programação 0-1 são incluídos na classe dos problemas NP-completos, para os quais não são conhecidos algoritmos de resolução polinomiais.

Por isso, grande esforço de pesquisa tem sido direcionado no sentido de desenvolver métodos capazes de resolver problemas grandes e complexos, dando origem ao que hoje se convencionou chamar de programação heurística.

Métodos heurísticos procuram encontrar *boas* soluções, o mais próximo possível das soluções ótimas, sem demandar um esforço computacional muito grande. Para tais métodos, não é garantida a otimalidade das soluções encontradas.

A seguir será apresentada uma revisão bibliográfica que inclui os principais enfoques empregados para solucionar problemas de programação 0-1 através de métodos heurísticos. Não há pretensão de ser exaustivo, mas simplesmente discutir sucintamente os métodos mais conhecidos e citados na literatura específica.

Nos anos 60 começaram a surgir os primeiros trabalhos com métodos heurísticos aplicados na resolução de problemas binários.

Um dos primeiros métodos heurísticos bem sucedidos que se tem notícia é devido a SENJU e TOYODA [24] (1968), que trata da resolução de problemas que buscam escolher, dentre um grande número de propostas independentes e indivisíveis, uma combinação de propostas de modo a chegar a um valor o mais próximo possível do objetivo desejado, satisfazendo as restrições impostas.

O problema resolvido por Senju e Toyoda é um caso especial do bem conhecido problema da mochila. Dentre m tarefas possíveis quer-se saber quais devem ser realizadas e quais não, sabendo-se que cada tarefa i realizada fornece um lucro relativo Ki e consome uma parcela Hij do recurso j.

A quantidade disponível de cada recurso j é Lj e há n recursos disponíveis. Deseja-se maximizar o lucro total.

Este problema pode ser formulado como:

$$\text{maximizar} \qquad Z = \sum_{i=1}^{n} K_i x_i$$

$$\text{sujeito a} \qquad \sum_{i=1}^{m} H_{ij}\, x_i \leq L_j \quad , \text{ para } j = 1,2,\ldots,n$$

$$x_i = 0 \text{ ou } 1 \quad , \text{ para } i = 1,2,\ldots,m$$

onde: $x_i = 1$, significa a decisão de realizar a tarefa i e $x_i = 0$, caso contrário.

Este caso especial do problema da mochila só admite coeficientes não negativos.

A estratégia usada pelo Método de Senju e Toyoda é bastante simples. O método inicia com uma solução ótima irrestrita (isto é, todas as variáveis com valor 1) e, com auxílio do conhecimento extraído do problema, vai trocando conjuntos de variáveis de 1 para 0 até obter uma solução factível. A escolha de qual variável deverá ser a próxima a ir de 1 para 0 depende do seu gradiente efetivo, que é o decréscimo da função objetivo por unidade de infactibilidade associada a cada variável ainda com valor 1.

Apesar da simplicidade, o método mostrou um desempenho satisfatório para uma variada gama de problemas, fornecendo soluções de boa qualidade e demandando pouco esforço computacional.

Um ano depois, HILLIER [19] (1969) propôs um método que resolvia o mesmo modelo de problema encontrado em Senju e Toyoda, acrescentando o tratamento de funções objetivo tanto de maximização quanto de minimização, com a ressalva que ele não podia tratar com restrições de igualdade, nem implicita, nem explicitamente.

O algoritmo de Hillier aplica uma estratégia com 3 fases distintas. Na primeira fase o processo tenta identificar uma região promissora, determinando uma solução ótima não inteira (obtida, por exemplo, com o auxílio de um algoritmo de resolução de problemas de programação linear) e um ponto cuja solução inteira aproximada não viole as restrições previamente impostas. Durante a segunda fase, o algoritmo busca um segmento de reta entre os dois pontos (partindo do ótimo encontrado para o PL) no sentido de encontrar uma solução inteira factível melhor. Na terceira fase é feita uma tentativa de melhorar a solução factível obtida, trocando o valor de uma ou duas variáveis de cada vez.

Já em 1970, ROTH [22] desenvolveu um método para resolver o problema geral de programação linear em variáveis bivalentes:

$$\text{minimizar} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{sujeito a} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad \text{,para } i = 1,2,\ldots,m$$

$$x_j = 0 \text{ ou } 1 \quad \text{,para } j = 1,2,\ldots,n$$

O método de Roth faz uso de uma busca parcial e de uma busca aleatória feita sobre um espaço de 2^n vetores n-dimensionais.

A busca parcial parte de pontos aleatóriamente escolhidos, seguindo por uma seqüência de vizinhanças (conjunto de vetores solução) definidas em relação ao ponto de partida. Esta busca termina quando não houver mais nenhum movimento, dentro deste conjunto de vizinhanças, que minimize a função. Isto caracteriza, precisamente, um ótimo local.

Partindo deste ponto, a busca aleatória é aplicada no sentido de superar o valor da solução obtida pela busca parcial que, provavelmente, não é um ótimo global. A busca aleatória faz uso de uma função chamada de *Função Sucessor*, que busca soluções que satisfaçam uma série de restrições, sempre tendo como objetivo a superação do ótimo local previamente encontrado.

O efeito combinado destes dois tipos de busca produz um

conjunto de soluções localmente ótimas, com boa probabilidade de conter o ótimo global.

O problema deste método é fazer um balanço efetivo entre a probabilidade do ótimo global estar contido no conjunto de soluções localmente ótimas geradas e o tempo computacional gasto para encontrar estas soluções. A principal dificuldade de trabalhar com este método está na descoberta deste balanço conveniente, que difere bastante de problema para problema.

Uma outra contribuição aos métodos heurísticos ocorreu com WYMAN [27], que em 1975 mostrou que programação inteira heurística poderia ser estudada com o propósito de, baseado nas propriedades computacionais das técnicas de programação inteira, desenvolver um conjunto de regras de decisão que auxiliasse o usuário a escolher entre técnicas ótimas e heurísticas.

Wyman fez um estudo comparativo entre um código ótimo para resolução de problemas 0-1, o RIP30C (ver GEOFFRION [9]) e a técnica heurística proposta por Senju e Toyoda, procurando estabelecer regras de escolha de um método ou outro, baseadas num dos critérios mais comuns usados em programação matemática: o esforço computacional em conjunto com o valor da função objetivo.

A comparação é feita variando-se 3 parâmetros da estrutura do problema: número de restrições, número de variáveis e severidade das restrições, sobre um total de 180 problemas. Foram propostos,também, modelos de regressão aplicados aos dados

com resultados promissores. Desenvolveram-se tabelas de desempate, para que o usuário, tendo em mãos o tamanho do problema, o esforço computacional e a função objetivo, pudesse rapidamente determinar a preferência por uma técnica ótima ou heurística.

Os resultados obtidos mostraram que as soluções heurísticas foram praticamente idênticas às soluções ótimas (na média, o valor encontrado pela heurística ficou a 98,6% do valor ótimo) sendo que elas requereram apenas 1/10 do esforço computacional (na média, 9,34% do esforço computacional requerido pelo método exato). E o que mais impressiona é o fato do algoritmo de Senju e Toyoda poder ser programado com menos de 50 instruções da linguagem FORTRAN (ver CÔTÉ e LAUGHTON [5]).

Estes resultados possibilitaram a Wyman dizer, na época, que a heurística iria eventualmente dominar qualquer técnica ótima para problemas razoavelmente difíceis.

O método proposto por KOCHENBERGER, McCARL e WYMAN [20], baseia-se no método de Senju e Toyoda, diferindo do mesmo em três aspectos importantes:

1) Inicia com um vetor solução com todos os elementos iguais a zero e incrementa as variáveis, baseado na quantidade de melhora que elas poderiam produzir na função objetivo por unidade de factibilidade usada;

2) Não coloca nenhuma restrição de sinal em qualquer dos coeficientes do problema;

3) Não restringe as variáveis a serem binárias.

A utilização deste método em programação 0-1 é imediata.

O método de Kochenberger, McCarl e Wyman foi testado em 26 problemas assim divididos: 9 problemas de alocação, 10 problemas de *fixed charge*, 6 problemas de *set covering* e um problema combinatório. Os resultados mostraram um desempenho médio de 86,3% em relação aos valores ótimos dos problemas testados. O método se mostrou mais eficiente nos problemas de *fixed charge* onde ficou, na média, a 94,4% do ótimo e apresentou uma maior dificuldade nos problemas de *set covering* ficando, na média, a 70% do valor ótimo.

Com o objetivo de resolver problemas de maior porte, TOYODA [25] fez algumas sofisticações no algoritmo que havia desenvolvido em conjunto com Senju e propôs um método simples e rápido com o objetivo de obter soluções práticas para problemas reais.

O método proposto é usado para obter uma solução aproximada para o problema multidimensional da mochila e não usa enumeração. Inicia com todas as variáveis em 0 e cada proposta é avaliada em função de seu gradiente efetivo, uma medida do seu

valor relativo projetado. O método é chamado de Gradiente Efetivo Primal (pois parte de uma solução factível) no qual, o gradiente efetivo é calculado para cada variável, levando-se em conta um vetor de penalidades que auxilia na escolha de qual variável será a próxima a trocar seu valor de 0 para 1. Ou seja, após o gradiente efetivo ser calculado, aplica-se um vetor de penalidades (que é diretamente proporcional ao coeficiente de custo da variável na função objetivo) a este gradiente e depois escolhe-se a variável com maior valor de gradiente efetivo penalizado para ser a próxima a trocar de valor. O método repete estes passos até que não exista mais nenhuma variável que possa trocar seu valor de 0 para 1, quando então ele pára e exibe a melhor solução encontrada até o momento.

O esforço computacional é relativamente pequeno mesmo para resolver problemas com mais de 1000 variáveis 0-1. O método produz boas soluções aproximadas para o problema multidimensional da mochila. Em adição, este método tem como característica atraente poder achar não somente a combinação ótima (ou quase ótima) de decisões, mas também poder avaliar a lucrabilidade de uma decisão de cada vez.

Este método é também aplicável a uma grande variedade de problemas binários de decisão (SIM - NÃO).

FERLAND e FLORIAN [7] desenvolveram um algoritmo sub-ótimo para resolver um problema 0-1 que aparece quando se aplica a decomposição de Benders. Ele produz uma boa solução

factível juntamente com a informação de um limitante superior da distância em relação à solução ótima. O algoritmo é bastante simples, pois envolve somente um processo de análise sobre cada bloco de variáveis (um de cada vez, mantendo fixos os restantes) e a aplicação de um procedimento simples para reduzir o valor da função objetivo. O processo é repetido com diferentes soluções iniciais, geradas a partir de informações sobre os custos relativos das variáveis.

O processo de análise determina um *falso mínimo local* do problema, associado com uma solução inicial factível. O limitante da distância deste *falso mínimo local* em relação à solução ótima é determinado com o auxílio da teoria de dualidade. A partir daí faz uma análise entre duas estratégias possíveis para sair deste *falso mínimo local* e reiniciar o processo de análise com uma nova solução factível inicial.

O método de Ferland e Florian foi testado em uma bateria de problemas, chegando sempre a menos de 4% do valor das soluções ótimas, com tempos computacionais de resolução bastante razoáveis. Ele se caracteriza como um bom método para ser usado na decomposição de Benders, onde não há necessidade de se resolver, a cada passo, o problema mestre relaxado até a otimalidade.

ZANAKIS [28] elaborou um estudo comparando o desempenho de três métodos heurísticos: Senju e Toyoda; Kochenberger, McCarl e Wyman; e Hillier, quando aplicados a problemas de programação

0-1 com coeficientes não negativos. Foi feita uma avaliação da eficácia dos métodos, medida em função do esforço computacional, erro e erro relativo sobre um conjunto de problemas da literatura e problemas teste 0-1 gerados aleatóriamente.

Análise de variância e regressão foram aplicadas para estudar o efeito do número de variáveis, número de restrições e o grau de severidade das restrições, que é medido com o auxílio da expressão (tomando-se por base a notação apresentada no modelo de problema encontrado em Roth):

$$S = \frac{\sum_{i=1}^{m} \left(\frac{bi}{\sum_{j=1}^{n} aij} \right)}{m}$$

Um resultado curioso observado, foi que quanto maior o número de variáveis, maior a exatidão dos métodos.

As diferenças de erros relativos entre os três métodos foram significantes (1:0,8:0,2) ou seja, se o método de Senju e Toyoda está a 1% do ótimo, o método de Kochenberger, McCarl e Wyman está a 0,8% do valor ótimo e o de Hillier a 0,2% do valor ótimo. Ainda que as diferenças entre os erros relativos dos três métodos fossem significativas, o erro relativo total ainda se manteve pequeno (menos que 2% na média) para a maioria das situações práticas.

O algoritmo de Hillier foi o mais exato, contudo foi muito lento e o que mais espaço, em termos computacionais, necessitou em relação aos outros dois métodos; o que faz dele o método menos indicado para problemas 0-1 grandes. A heurística de Kochenberger, McCarl e Wyman, foi a mais rápida e também a mais exata das três, para problemas pouco restritos. Em geral o algoritmo de Senju e Toyoda foi o mais rápido, mas menos exato que os outros em problemas de tamanho médio e pequeno.

Zanakis também sugere a melhor heurística a ser usada, baseado nas características do problema e ainda diz que se não for exigida uma solução ótima, muitas vezes é melhor usar os métodos de Senju e Toyoda e de Kochenberger, McCarl e Wyman e depois escolher a melhor solução entre os dois ao invés de se usar um método exato.

Pesquisas mais recentes têm buscado desenvolver heurísticas de *desempenho garantido*, ou seja, heurísticas que garantam estar dentro de certos limites de proximidade do ótimo, com um esforço computacional que seja função polinomial dos valores de entrada do problema. Esta classe de heurísticas tem encontrado dificuldades quando usadas na análise do pior caso:

1) A proximidade do ótimo que atenda a um tempo polinomial é, geralmente, ruim (longe do ótimo real) para propósitos práticos;

2) Geralmente não existe nenhuma correlação entre a análise do desempenho médio e a do pior caso.

BALAS, que foi um dos precursores na resolução de problemas 0-1, publicando em 1965 [3] um algoritmo exato de enumeração implícita para este tipo de problema, se uniu a MARTIN [4] para desenvolver uma heurística que é conhecida como *Pivot e Complemento*. Trata-se de uma heurística para problemas 0-1, sem atrativos matemáticos, mas que trabalha muito bem, tanto do ponto de vista do esforço computacional envolvido quanto da qualidade das soluções obtidas.

O algoritmo está baseado no fato que um programa 0-1 do tipo:

$$(P) \quad \text{Max} \quad cx$$
$$\text{s.a.} \quad Ax \leq b$$
$$x_j = 0 \text{ ou } 1 \qquad , \ j \in N$$

onde: A é uma matriz mxn

b é um vetor mx1

$N = \{ 1,\ldots,n \}$

$M = \{ 1,\ldots,m \}$

é equivalente ao problema:

$$(P') \quad \text{Max} \quad cx$$
$$\text{s.a.} \quad Ax + y = b$$
$$0 \leq x \leq e$$
$$y \geq 0$$
$$y_i \text{ seja básica} \quad \forall i \in M$$

onde: $e = (1,\ldots,1)$ tem n componentes.

A heurística Pivot e Complemento inicia resolvendo o problema linear (PL) obtido com a relaxação das restrições $x_j = 0$ ou 1, $\forall j \in N$.

A partir do tableau ótimo do problema linear relaxado ele faz uma seqüência de pivoteamentos, objetivando colocar na base todas as variáveis de folga y_i com um custo mínimo na factibilidade dual, enquanto, ocasionalmente elimina infactibilidades primais pela complementação de algumas variáveis. Complementar uma variável x_j, significa mover x_j de um de seus limites até o limite oposto. Esta fase é chamada *Fase de Busca* de uma solução 0-1 factível.

Numa outra fase, chamada de *Fase de Melhoramento* da solução 0-1 atual, o algoritmo faz uma busca local, novamente baseada na complementação de certas variáveis e procurando melhorar o valor da função objetivo.

De forma geral é possível afirmar que os melhores métodos heurísticos propostos para "resolver" problemas 0-1 têm conseguido atrair atenção e respeito devido, principalmente, à feliz combinação de duas características: as soluções aproximadas que produzem são de excelente qualidade e os tempos computacionais para produzir tais soluções são expressivamente pequenos quando comparados com os tempos gastos pelos métodos exatos.

A experiência computacional de Balas e Martin ilustra estas afirmações. A heurística proposta pelos autores foi testada em 92 problemas com soluções ótimas conhecidas. Uma parte deles é constituída de problemas reais, retirados da literatura e outra parte foi gerada aleatóriamente. O tamanho dos problemas varia de 5 a 200 restrições e de 20 a 900 variáveis. A heurística achou soluções ótimas para 45 dos 92 problemas, e para 81 deles ficou sempre abaixo de 1% das soluções ótimas correspondentes.

Em média um problema demorou 5,73 segundos para ser resolvido, e o problema mais demorado consumiu 50 segundos de CPU num UNIVAC 1108.

CROWDER, JOHNSON e PADBERG [6] mostram que algoritmos exatos para resolver problemas 0-1 de grande porte precisam incorporar diferentes estratégias matemáticas, com boa dose de sofisticação para produzir resultados em tempos satisfatórios.

O método de Crowder, Johnson e Padberg foi testado em 10 problemas com número de restrições variando de 16 a 756 e número de variáveis entre 33 e 2756, chegando às soluções ótimas em todos eles, num tempo médio de 618,48 segundos de CPU em um IBM 370/168 - MVS.

Heurísticas também são usadas de forma combinada com métodos exatos, a fim de melhorar o desempenho geral. A classe dos algoritmos enumerativos freqüentemente faz uso de heurísticas para melhorar o desempenho computacional, e muitos códigos

comerciais colocam à disposição o *modo heurístico* que trabalha até o usuário achar que a solução factível fornecida é satisfatória.

Uma contribuição recente para a solução de problemas combinatoriais é a de GLOVER [12] e [15]. Ele propõem a *Busca Tabu*, que pode ser vista como uma meta-heurística, super-imposta sobre outra heurística. Esta estratégia busca transceder a otimalidade local, proibindo ou, mais amplamente, penalizando certos movimentos. O intuito de classificar um movimento como proibido (ou, tabu) está direcionado no sentido de evitar ciclagem e exploração de soluções que já foram pesquisadas.

A busca tabu vem sendo aplicada com sucesso em vários problemas de otimização combinatória e parece ser um veio promissor para a busca de soluções de qualidade, em tempos computacionais razoáveis, para problemas grandes e complexos.

O objetivo principal desta tese é explorar o uso de busca tabu em programação 0-1. É reconhecida a impotência das heurísticas frente aos ótimos locais. Elas trabalham até que melhorias não sejam mais possíveis. A partir daí, é preciso abandonar esta direção de busca e reinicializar a heurística, o que, em geral, não produz bons resultados. Uma outra alternativa é alargar o seu espectro de busca, o que a torna mais demorada. Poder-se-ia mesmo dizer que o bom desempenho de uma heurística está relacionado não só com a sua habilidade em chegar rápido em ótimos locais, mas também saber sair deles de forma inteligente.

Isto nos leva a idéia de incorporar as estratégias da busca tabu em uma heurística de programação 0-1, de modo a dotá-la de mecanismos inteligentes capazes de evitar as armadilhas dos ótimos locais e, com isto, alcançar soluções melhores.

Estas idéias não são novas. Desde o aparecimento da busca tabu, inúmeras aplicações de sucesso foram publicadas [10], [11],[12], [14], [16], [17], [18] e [26]. O objetivo central é verificar o comportamento da busca tabu em problemas lineares do tipo 0-1.

No próximo capítulo será mostrado como se adapta a busca tabu à heurística de Balas e Martin. Para facilitar o acesso aos fundamentos da busca tabu é apresentado um resumo no Apêndice. A leitura do apêndice facilita a compreensão do próximo capítulo, mas torna-se dispensável, caso se conheçam os rudimentos da busca tabu.

CAPÍTULO 2

INCLUSÃO DE BUSCA TABU EM UMA HEURÍSTICA DE PROGRAMAÇÃO BINÁRIA

O objetivo desta tese é incorporar os procedimentos da busca tabu em um algoritmo desenvolvido para resolver problemas de programação 0-1. Mais especificamente, pretende-se estudar os eventuais ganhos que a aplicação de busca tabu consegue introduzir em uma heurística clássica para esta classe de problemas.

A primeira etapa consiste em escolher uma heurística apropriada para permitir a aplicação pretendida. A escolha recaiu sobre a heurística de pivot e complemento de BALAS e MARTIN [4], primeiramente, por ser um dos mais eficientes métodos heurísticos já desenvolvidos para problemas 0-1 e depois porque a estratégia de busca de melhores soluções é baseada em processos de trocas (complementos) de variáveis, que têm estrutura bastante conveniente para as aplicações das técnicas da busca tabu.

A seguir será descrito o método de Balas e Martin e,

conjuntamente, se introduzirá busca tabu em sua estrutura original.

A classe de problemas que trata este trabalho pode ser formulada como:

$$(P) \qquad \max \quad cx$$
$$\text{s.a.} \quad Ax \leq b$$
$$x_j = 0 \text{ ou } 1, \quad j \in N$$

onde: A é uma matriz $m \times n$;

b é um vetor $m \times 1$;

c é um vetor $1 \times n$;

$N = \{ 1, 2, \ldots, n \}$;

$M = \{ 1, 2, \ldots, m \}$.

O método de Balas e Martin está fundamentado no fato de que o problema (P) é equivalente a:

$$(P') \qquad \max \quad cx$$
$$\text{s.a.} \quad Ax + y = b$$
$$0 \leq x \leq e$$
$$y \geq 0$$
$$y_i \text{ seja básica } \forall i \in M$$

onde: $e = (1, \ldots, 1)$ tem n componentes.

A forma relaxada de (P) pode ser escrita como:

$$(PL) \qquad \max \quad cx$$
$$\text{s.a.} \quad Ax \leq b$$
$$0 \leq x_j \leq 1 , \quad j \in N$$

O algoritmo inicia resolvendo o problema linear relaxado (PL) (assumindo que todo c_j é inteiro, para $j \in N$), isto é, encontrando uma solução básica ótima utilizando, por exemplo, o algoritmo primal simplex canalizado. A dimensão da base é m e as variáveis 0-1 não básicas podem estar em seu limite inferior (0) ou superior (1).

Portanto, em cada estágio, o tableau simplex pode ser representado por:

$$x_i = a_{io} + \sum_{j \in J} a_{ij} (-x_j) , \quad i \in \{ 0 \} \cup I$$

onde: I é o conjunto de índices das variáveis básicas;
J é o conjunto de índices das variáveis não básicas;
0 é o índice da linha da função objetivo.

O algoritmo é dividido em 2 fases distintas:

A primeira é chamada de FASE DE BUSCA e procura encontrar uma solução 0-1 factível, de valor razoável, realizando

pivoteamentos. Em alguns casos, também são utilizadas outras ações como complementação, arredondandamento ou truncamento de variáveis.

A segunda fase é chamada de FASE DE MELHORAMENTO e tenta melhorar a solução encontrada na primeira fase (caso a fase de busca não tenha falhado em encontrar uma), complementando certos conjuntos de variáveis.

Quando não houver mais nenhum complemento de variáveis, dentre o conjunto de complementos disponíveis, que produza algum incremento na melhor solução corrente (caracterizando um ótimo local), o algoritmo de Balas e Martin pára.

A partir daí, inicia-se a busca tabu, que se constitui numa terceira fase (chamada de FASE TABU), onde se busca a realização de movimentos que causem a mínima degradação no valor da solução corrente, procurando uma direção de saída do ótimo local para outras regiões onde se espera encontrar soluções melhores que a corrente.

2.1. DESCRIÇÃO GERAL DO ALGORITMO

Esta seção, inicialmente, apresenta e discute alguns conceitos e operações que serão utilizados na construção do algoritmo.

Pivot do Tipo 1 Um pivot do tipo 1 é aquele que mantém a factibilidade primal e troca uma variável de folga não básica por uma variável 0-1 básica, isto é, um pivoteamento na coluna $q \in J \setminus N$ [1] (variáveis de folga não básicas) e na linha $p \in N$, tal que:

$$\frac{a_{po}}{a_{pq}} = \min_{i \in I \cap N} \left\{ \min_{a_{iq}>0} \frac{a_{io}}{a_{iq}} \; , \; \min_{a_{iq}<0} \frac{(1 - a_{io})}{|a_{iq}|} \right\}$$

Pivot do Tipo 2 Um pivot do tipo 2 é aquele que mantém a factibilidade primal e não muda o número de variáveis básicas 0-1 (isto é, troca uma variável de folga por outra variável de folga ou uma variável estrutural por outra variável estrutural). Além disso busca reduzir a soma das infactibilidades inteiras, definida por

$$\sum_{i \in I \cap N} \min \{ a_{io} , (1 - a_{io}) \} ,$$

de no mínimo um Δ.

Pivot do Tipo 3 Um pivot do tipo 3 é aquele que troca uma variável de folga não básica por uma variável 0-1 básica, sacrificando a factibilidade primal. A restrição imposta é que a variável de folga entre na base com um valor positivo.

[1] $A \setminus B$ significa uma operação entre conjuntos, cujo resultado é um outro conjunto compreendendo os elementos do conjunto A excluindo os elementos do conjunto B pertencentes a A.

Complementos Complementar uma variável x_j significa mover x_j de um de seus limites até o outro.

Complementos simples, duplos ou triplos, correspondem a complementar uma, duas ou três variáveis de uma só vez.

Na *fase de busca* as variáveis são complementadas para reduzir a medida de infactibilidade, definida por:

$$\sum_{i \in I} \max \{ 0 , - a_{io} \}.$$

Um conjunto $S \subseteq J \cap N$ de variáveis não básicas, com $|S| = 1$ ou 2 (onde $|\cdot|$ representa o número de elementos do conjunto), é candidato a complementação se:

$$\sum_{i \in I} \max \{ 0 , - a_{io} \} - \sum_{i \in I} \max \{ 0 , - a_{io} + \sum_{j \in S} a_{ij} \} \geq \Delta > 0$$

Na *fase de melhoramento*, as variáveis devem ser complementadas para melhorar o valor da solução 0-1 atual. Sendo:

N^+ = conjunto das variáveis 0-1, atualmente em seu limite inferior

N^- = conjunto das variáveis 0-1, atualmente em seu limite superior

define-se:

$$d_j = \begin{cases} c_j, & j \in N^+ \\ -c_j, & j \in N^- \end{cases} \qquad f_{ij} = \begin{cases} a_{ij}, & j \in N^- \\ -a_{ij}, & j \in N^+, \end{cases} \quad \text{para } i=1,2,\ldots,m$$

Um conjunto $S \subseteq N$, com $|S| = 1$, 2 ou 3 é candidato a complementação se:

$$\sum_{j \in S} d_j \geq 1 \quad ; \quad \sum_{j \in S} f_{ij} \leq b_i - \sum_{j \in N^+} a_{ij}, \quad \text{para } i = 1,2,\ldots,m$$

e o complemento não for considerado proibido.

Na *fase tabu* a complementação de uma variável causa uma piora no valor da solução 0-1 atual.

Usando a mesma notação apresentada para complementos na fase de melhoramento, temos que um conjunto $S \subseteq N$, com $|S| = 1$, 2 ou 3 é candidato a complementação se:

$$\sum_{j \in S} d_j \leq 0 \quad ; \quad \sum_{j \in S} f_{ij} \leq b_i - \sum_{j \in N^+} a_{ij}, \quad \text{para } i = 1,2,\ldots,m$$

e o complemento não for considerado proibido.

Fixação de Variáveis Considerando:

Z o valor da função objetivo da solução 0-1 atual;

Z_{PL} o valor da função objetivo na solução ótima do (PL);

$\bar{c}_j$ o custo relativo da variável $j \in N$ no (PL) ótimo, isto é : $\bar{c}_j \geq 0$ se x_j é não básica em 0 (limite inferior) e $\bar{c}_j \leq 0$ se x_j é não básica em 1 (limite superior).

Tem-se que se $|\bar{c}_j| > Z_{PL} - Z - 1$, para algum $j \in N$, então, em qualquer solução melhor que a atual, o valor da variável x_j deve ser o mesmo que ela tinha na solução ótima do (PL) e se $|\bar{c}_j| > Z_{PL} - Z$, então x_j tem este mesmo valor na solução atual.

Portanto, sempre que for encontrada uma variável 0-1 que satisfaça uma das seguintes condições:

(i) tenha um custo relativo que exceda $Z_{PL} - Z$;

(ii) tenha um custo relativo que exceda $Z_{PL} - Z - 1$ e tenha o mesmo valor tanto na solução atual quanto na solução ótima do (PL),

então é possível fixar a variável naquele valor até o fim do procedimento.

Fica claro que, se $Z_{PL} - Z - 1 \leq 0$, o valor da solução

0-1 atual é ótimo e todas as variáveis estão fixadas em seus valores ótimos.

Arredondamento e Truncamento de Soluções Em certos estágios do algoritmo procura-se encontrar uma solução 0-1 factível arredondando ou truncando a solução atual (fracionária).

Arredondar uma solução, significa arredondar todas as variáveis 0-1 fracionárias para o seu valor inteiro mais próximo.

Truncar uma solução, significa substituir todos os valores fracionários das variáveis 0-1 por 0.

Movimento Proibido Antes de caracterizar um movimento proibido, primeiro é conveniente mostrar como um movimento proibido é armazenado.

Proibir um movimento significa colocá-lo numa lista de tamanho fixo, que será inspecionada cada vez que se quiser checar a validade de um movimento.

Para isto são criadas as *listas tabu*, uma para cada tipo de complemento, ou seja:

- para complementos simples ⇒ TABULIST1;
- para complementos duplos ⇒ TABULIST2;
- para complementos triplos ⇒ TABULIST3.

As listas são sempre inicializadas com todos os seus componentes tendo valores nulos e cada lista pode ter tamanho diferente, dependendo da estrutura de cada problema.

Supondo, por exemplo, que o tamanho escolhido para TABULIST1 seja t e sendo s_k o k-ésimo movimento, do tipo complemento simples, que acabou de ser realizado, a TABULIST1 é representada por:

$$TABULIST1 = \{ s_h : h > k - t \}$$

Ou seja, a lista tabu armazena sempre os t últimos movimentos realizados.

Deve-se notar que cada tipo de movimento deve ter um contador de iterações independente. Outro ponto relevante é que a lista guarda o movimento direto e não o seu reverso.

Uma consideração a ser feita é que movimento direto é aquele que causa uma *melhora* na solução atual e movimento reverso é aquele que causa uma *piora* na solução.

A entrada de um movimento na lista é gerenciada através da expressão:

$$p = k - \left\lceil \frac{k}{t} \right\rceil t$$

onde: p é a posição em que o movimento k entra na lista e

$\lceil \cdot \rceil$ significa o maior inteiro menor ou igual.

Convenciona-se que se p = 0 com k > 0, então faz-se p = t.

Os movimentos são guardados dentro das listas tabu através de um par de *atributos* (VAR ; VALOR(VAR)).

VAR → indica o número da variável que foi complementada no movimento realizado. Caso se trate de um movimento duplo ou triplo guardam-se os números das variáveis complementadas.

VALOR(VAR) → - no caso de movimento direto é o valor 0 ou 1 que a variável (ou variáveis) assumiu após a complementação;

- no caso de movimento reverso é o valor 0 ou 1 que a variável (ou variáveis) tinha *antes* da complementação.

Se, ao tentar realizar um movimento de um determinado tipo, for verificado que seus atributos coincidem com o par de atributos de sua lista tabu correspondente, então este movimento é considerado proibido e é descartado.

A escolha destes atributos para as listas tabu, foi orientada no sentido de se trabalhar com proibições bem simples e bastante restritivas, constituindo um referencial básico para comparações posteriores com esquemas menos restritivos e mais informados.

Movimento Possível Um movimento possível é aquele que além de não ser considerado proibido deve gerar uma solução 0-1 factível. Se estas condições forem satisfeitas, o movimento é um candidato a complementação.

Movimento Reverso Quando a fase de melhoramento não tem mais sucesso em encontrar complementos simples, duplos ou triplos que melhorem a solução 0-1 corrente, o algoritmo tenta realizar um movimento reverso para que o procedimento, partindo de outra solução, possa encontrar soluções melhores que a atual.

Tem-se sempre em mente buscar o movimento reverso que cause a menor degradação possível no valor da solução atual (lembrando que o melhor movimento reverso é aquele feito num conjunto de variáveis cujo decremento causado na solução atual seja nulo).

Critérios de Parada Como se está diante de um algoritmo heurístico com a capacidade de transcender ótimos locais, pode ocorrer dele nunca parar, ou entrar em um ciclo contínuo, caso não se colocaquem limitantes em sua execução. Por

isto são sugeridos *critérios de parada* para o algoritmo, alguns imediatos, outros não. Os critérios de parada são os seguintes:

(1) Se todas as listas tabu estiverem vazias, PARE. Isto acontece quando a solução encontrada ao fim da fase de busca é a melhor solução encontrada pelo algoritmo, então não se tem nenhum movimento realizado na fase de melhoramento, e portanto as listas tabu estão vazias;

(2) Nenhum movimento reverso é possível. Isto acontece quando o algoritmo entra na fase tabu e, ou não há nenhum movimento candidato a reverso possível ou, dentre os possíveis, nenhum movimento leva à uma solução 0-1 factível; se isto ocorrer, PARE.

(3) O número de iterações chegou ao máximo. Como foi dito anteriormente, o algoritmo deve ter um limitante para que o procedimento não se torne um ciclo contínuo; portanto é colocado um limite para o número de iterações, onde se entende número de iterações pela soma dos movimentos diretos e reversos simples, duplos e triplos já realizados. Se este número de iterações exceder seu limite, PARE.

Com isso o algoritmo torna-se finito, e, caso tenha passado pela fase de melhoramento ao menos uma vez, exibirá a melhor solução 0-1 encontrada até aquele momento, que poderá ser a ótima ou uma solução o mais próximo possível dela.

A seguir é apresentada a versão inicial, passo a passo, do algoritmo desenvolvido, já incluindo busca tabu em sua estrutura.

2.2. ALGORITMO DE PROGRAMAÇÃO BINARIA INCLUINDO BUSCA TABU

O algoritmo é dividido em 3 fases e consiste na seguinte estrutura:

FASE DE BUSCA

Passo 1 Resolva o (PL). Se a solução for inteira PARE, ela é ótima.
Caso contrário, vá para o Passo 2.

Passo 2 Busque um pivot do tipo 1, se não existir vá para o Passo 3.
Caso contrário, execute o pivot do tipo 1 que resulte no maior valor da função objetivo e vá para o Passo 4.

Passo 3 Busque um pivot do tipo 2, se não existir vá para o Passo 5.
Caso contrário, execute o primeiro pivot do tipo 2 encontrado e vá para o Passo 4.

Passo 4 Examine a solução básica atual, se ela é inteira vá para a FASE DE MELHORAMENTO.

Caso contrário, vá para o Passo 2.

Passo 5 Verifique se arredondando ou truncando a solução básica atual, produz-se uma solução 0-1 factível. Se sim, vá para a FASE DE MELHORAMENTO.

Caso contrário, vá para o Passo 6.

Passo 6 Execute um pivot do tipo 3 que minimize o valor da medida de infactibilidade e vá para o Passo 7.

Passo 7 Busque uma única variável 0-1, não básica, cuja complementação reduz o valor da medida de infactibilidade. Se não existir nenhuma, vá para o Passo 9.

Caso contrário, complemente a variável que produz a maior redução na medida de infactibilidade e vá para o Passo 8.

Passo 8 Se a solução atual é infactível, vá para o Passo 7.

Se ela é factível, verifique se arredondando ou truncando a solução atual produz-se uma solução 0-1 factível. Se sim, vá para a FASE DE MELHORAMENTO. Caso contrário, vá para o Passo 2.

Passo 9 Busque um par de variáveis 0-1, não básicas, cuja complementação reduz o valor atual da medida de infactibilidade. Se não houver nenhum, PARE, o procedimento heurístico FALHOU (não encontrou nenhuma solução 0-1 factível).

Caso contrário, complemente o primeiro par identificado e vá para o Passo 8.

FASE DE MELHORAMENTO

Passo 1 Fixe todas as variáveis 0-1 que possam ser fixadas e vá para o Passo 2.

Passo 2 Busque uma única variável 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um incremento na solução 0-1. Se não existir vá para o Passo 3.

Caso contrário, complemente a variável que produz a maior melhora na solução 0-1; proiba o movimento direto e o seu reverso e vá para o Passo 1.

Passo 3 Busque um par de variáveis 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um incremento na solução 0-1. Se não existir vá para o Passo 4.

Caso contrário, complemente o primeiro par candidato, proiba o movimento direto e o seu reverso e vá para o Passo 1.

Passo 4 Busque um trio de variáveis 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um incremento na solução 0-1. Se não existir vá para a FASE TABU.

Caso contrário, complemente o primeiro trio candidato, proiba o movimento direto e o seu reverso e vá para o Passo 1.

FASE TABU

Passo 1 Teste os critérios de parada do algoritmo. Se algum for satisfeito, exiba a melhor solução 0-1 encontrada até o momento e PARE.

Caso contrário, vá para o Passo 2.

Passo 2 Busque uma única variável 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um decremento na solução 0-1. Se não existir, vá para o Passo 3.

Caso contrário, complemente a variável que produz a menor piora na solução 0-1, proiba o movimento direto e o seu reverso e vá para a FASE DE MELHORAMENTO.

Passo 3 Busque um par de variáveis 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um decremento na solução 0-1. Se não existir, vá para o Passo 4.

Caso contrário, complemente o primeiro par candidato, proiba o movimento direto e o seu reverso e vá para a FASE DE MELHORAMENTO.

Passo 4 Busque um trio de variáveis 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um decremento na solução 0-1. Se não existir, vá para o Passo 1.

Caso contrário, complemente o primeiro trio candidato, proiba o movimento direto e o seu reverso e vá para a FASE DE MELHORAMENTO.

A seguir, discutem-se alguns aspectos computacionais da implementação do algoritmo e também alguns resultados desta versão do algoritmo, obtidos sobre um conjunto de problemas teste.

2.3. ASPECTOS E RESULTADOS COMPUTACIONAIS

Na implementação do algoritmo usou-se a linguagem de programação PASCAL-SVS, incluindo algumas rotinas do sistema operacional, como por exemplo, a rotina de contagem do tempo de CPU gasto na execução do programa.

A máquina usada foi um mini-computador DIGIREDE 8000/2 que trabalha em um ambiente UNIX multi-usuário, gerenciado pelo Sistema Operacional DIGIX.

Os resultados apresentados na seqüência foram obtidos pela execução do algoritmo como foi anteriormente apresentado. O algoritmo utilizado na resolução do problema linear relaxado (PL) (Passo 1 da Fase de Busca do algoritmo) foi um algoritmo primal simplex canalizado em sua forma simples (não revisada) (ver GARFINKEL e NEMHAUSER [8]). O motivo de usar este algoritmo foi o fato de ter à disposição o tableau simplex completo, o que facilita a aplicação da fase de busca do algoritmo heurístico. Devido a isto, muitas operações desnecessárias são realizadas, o que, em conjunto com o algoritmo sem sofisticações apresentado, leva a tempos computacionais não muito bons.

Apesar disto a aplicação da busca tabu no processo levou a soluções, na grande maioria das vezes, de qualidade superior àquelas encontradas por Balas e Martin, o que motivou uma continuação do trabalho no sentido de, mantendo a estrutura tabu, diminuir o tempo e o esforço computacional necessários para a resolução de problemas maiores e mais difíceis. O resultado deste esforço adicional será discutido no próximo capítulo.

Dos problemas que foram resolvidos, uma parte é encontrada na literatura e a outra foi gerada aleatóriamente.

Os problemas da série PETER foram obtidos de PETERSEN [21].

Já os problemas da série CEM e DUZ foram gerados aleatóriamente do seguinte modo. Os coeficientes c_j são números aleatórios inteiros entre 0 e 99, cada a_{ij} é um número aleatório entre 0 e 99 multiplicado por c_j / 10, enquanto b_i é um número aleatório entre 5 e 9, multiplicado por s_i / 10, onde $s_i = \sum_j a_{ij}$. Esta geração aleatória produz problemas de *capital budgeting* um pouco mais difíceis e também mais realistas do que aqueles em que todos os coeficientes são gerados aleatóriamente, sem nenhuma relação entre o c_j e seus a_{ij} correspondentes.

Os problemas da série P são gerados de forma semelhante aos da série CEM e DUZ, diferindo nos seguintes aspectos. Os coeficientes c_j são números inteiros entre 1 e 100, cada a_{ij} é

gerado da mesma maneira anterior e b_i é um número aleatório entre 3 e 5, multiplicado por s_i / 50. Os problemas da série P são considerados difíceis pois a severidade das restrições determina a existência de poucas soluções factíveis, dificultando a busca.

A tabela 2.1 mostra os resultados obtidos com esta versão inicial. Os resultados e comparações obtidas com o algoritmo melhorado serão apresentados no capítulo seguinte.

DADOS DO PROBLEMA			DADOS RELATIVOS AO DESEMPENHO DO ALGORITMO							DADOS RELATIVOS A TEMPOS DE CPU			
NOME	n	m	Zot	(Zot-Zpc)/Zot (x100%)	(Zot-Zlb)/Zot (x100%)	FIX	ITER	REV	MSOL	Tt (seg.)	Tpc/Tt x100%	Tpl/Tt x100%	Tlb/Tt x100%
PETER3	15	10	4015	17,186	0,000	3	52	11	20	7,8	6,4	10,3	83,3
PETER4	20	10	6120	0,927	0,000	12	50	11	39	5,4	12,3	14,0	73,7
PETER5	28	10	12400	0,161	0,000	15	54	11	35	15,8	5,7	11,1	59,2
PETER6	39	5	10618	0,283	0,283	6	57	11	10	69,4	1,6	18,2	80,2
PETER7	50	5	16597	0,230	0,097	13	59	11	13	124,4	3,5	5,0	91,5
CEM53	100	5	3472,51	0,504	0,504	43	50	15	5	491,4	1,6	8,2	90,2
CEM54	100	5	3579,57	0,156	0,156	78	52	15	2	41,9	11,2	4,9	84,5
DUZ51	200	5	6795,86	0,086	0,086	136	53	16	4	1324,5	1,8	7,0	91,3
DUZ59	200	5	6247,44	0,167	0,151	104	51	15	12	2984,3	1,1	22,2	76,7
CEM106	100	10	3329,71	0,502	0,442	29	50	14	23	1077,8	1,3	5,2	93,5
CEM108	100	10	3112,64	0,470	0,470	35	50	14	5	801,8	2,3	5,7	92,0
DUZ101	200	10	5998,90	0,665	0,665	5	53	13	12	41678,3	0,2	17,1	82,7
DUZ104	200	10	6602,66	0,192	0,146	91	52	13	12	3940,9	2,1	8,0	89,9
P2	50	35	167,449	20,528	28,091	1	53	12	21	269,7	19,2	10,0	70,8
P3	100	30	750,087	7,744	7,944	0	54	12	18	9405,8	1,7	8,8	89,2
P4	200	30	1513,14	4,305	4,305	1	51	15	11	85947,2	0,3	5,9	93,8

TABELA 2.1 - DESEMPENHO DO ALGORITMO TABU SEM MELHORIAS

onde:

n - número de variáveis 0-1 do problema;

m - número de restrições do problema;

Zot - nas séries PETER, corresponde ao valor da solução ótima do problema 0-1;

nas séries CEM, DUZ e P, corresponde ao valor da solução ótima do problema linear relaxado;

Zpc - valor de solução 0-1 encontrado pela heurística pivot e complemento (Balas e Martin);

Ztb - valor de solução 0-1 encontrado com a aplicação de busca TABU;

FIX - número de variáveis fixadas no processo;

ITER - número de iterações realizadas;

REV - número de movimentos reversos realizados;

M SOL - iteração na qual foi encontrada a melhor solução 0-1;

Tt - tempo total de CPU gasto na resolução do problema;

Tpl - tempo gasto na resolução do (PL);

Tpc - tempo gasto até encontrar a melhor solução da heurística pivot e complemento;

Ttb - tempo gasto pela busca tabu na tentativa de melhorar a solução;

Tamanho de lista tabu igual a 7, para as três listas.

CAPÍTULO 3

ALGORITMO TABU COM MELHORIAS COMPUTACIONAIS

Neste capítulo são introduzidas mudanças de caráter computacional com a finalidade de diminuir o tempo de execução do algoritmo tabu. Para isto foram desenvolvidos alguns testes *inteligentes* adicionais, estruturas de dados especializadas e alguns algoritmos auxiliares que serão discutidos em seguida.

3.1. TESTES ADICIONAIS

Analisando os resultados dos testes computacionais realizados anteriormente, é possível observar que após uma certa combinação de movimentos diretos e reversos, houve casos onde os movimentos subsequentes levavam a soluções que já haviam sido visitadas antes, o que caracterizava uma ciclagem, pois o algoritmo tenderia a repetir os mesmos passos realizados anteriormente.

A fim de evitar esta busca inútil resolveu-se

restringir mais o espectro da busca através da inclusão de mais um par de atributos nas listas tabu. Ou seja, para um movimento candidato ser executado, ele precisava passar pelo par de atributos (VAR , VALOR(VAR)). Agora, ele também precisa passar pelo par (SOLUÇÃO , TIPO(SOLUÇÃO)), onde:

SOLUÇÃO - valor que assume a função objetivo na solução 0-1 após a complementação;

$$\text{TIPO (SOLUÇÃO)} = \sum_{i=1}^{n} x_i * i \quad , \quad i \in N.$$

Este número serve para diferenciar soluções distintas, mas com o mesmo valor de função objetivo.

Note que o movimento candidato só não será realizado se ele tiver o seu correspondente par (SOLUÇÃO , TIPO(SOLUÇÃO)) já presente em algumas das três listas tabu.

Outros dois testes adicionais foram acrescentados para aumentar o desempenho do algoritmo:

(1) as variáveis fixadas ao longo do algoritmo são retiradas das análises de futuras complementações.

(2) Caso o complemento candidato que está sendo analisado melhore o valor da solução atual, mas esta melhora é tal que ultrapassa o valor ótimo do PL, pode-se descartar este movimento, pois já se sabe que ele levará a uma solução

infactível.

3.2. ESQUEMA DE ORDENAÇÃO DAS VARIÁVEIS CANDIDATAS A COMPLEMENTAÇÃO

Seguindo o princípio básico da busca tabu de procurar realizar o *melhor movimento possível* a cada iteração, resolveu-se incorporar ao código computacional um algoritmo de ordenação das variáveis, com o objetivo de orientar melhor e mais rapidamente a escolha sobre quais variáveis complementar.

Para que esta adaptação fosse possível, restringiu-se os coeficientes de custo, anteriormente sem restrição de sinal, a assumirem somente valores não negativos, ou seja, $c_j \geq 0$ e inteiro $\forall j \in N$. Note que esta suposição não leva a uma perda de generalidade,pois com uma simples troca de variável, pode-se mudar o sinal do coeficiente de custo associado a uma variável do problema.

Para fazer uma ordenação eficiente usou-se uma adaptação do MÉTODO RADIX (ver AHO, HOPCROFT e ULLMAN [1] e [2]).

Para facilitar a compreensão do método de ordenação considere um exemplo com X = (1,0,1,1,1,0,0,1,1,0) sendo uma solução qualquer, V = (1,2,3,4,5,6,7,8,9,10) sendo o número das variáveis associadas ao vetor solução e

C = (13,66,66,0,21,54,79,84,22,80) o vetor de custo.

Como é necessário separar as variáveis que estão com valor 0 das que estão com valor 1, aproveita-se a rotina de ordenação e criam-se duas listas ordenadas, a das variáveis que estão em zero e a das variáveis que estão em um.

A rotina de ordenação inicia com a criação de um vetor que armazenará certos dados das variáveis, servindo como intermediário entre o vetor completamente desordenado e os vetores ordenados das variáveis em zero e em um. Este vetor pode ser expresso pelo seguinte modelo:

ARMAZENADOR (0 : RANGE , 1 : NUM_INT) = VAR
Coef. 1 Coef. 2 Coef. 3

onde:

Coef. 1 : é o índice correspondente ao valor do coeficiente de custo da variável em questão.
RANGE - corresponde ao valor máximo do coeficiente de custo da função objetivo acrescido de uma unidade (neste caso RANGE = 85).

Coef. 2 : é usado para o caso de haver coeficientes de custo repetidos; então a primeira aparição de um determinado valor de custo terá o valor de Coef. 2 igual a 1, a

segunda aparição deste mesmo valor de custo terá um Coef. 2 igual a 2 e assim por diante.

NUM_INT - corresponde ao número de variáveis 0-1 do problema, que é o valor máximo que este índice pode assumir (no caso de todos os coeficientes de custo terem o mesmo valor), neste exemplo NUM_INT = 10.

Coef. 3 : corresponde ao número da variável associada ao coeficiente de custo em questão.

Inicialmente tem-se que todos os elementos do vetor estão no valor zero.

Então percorre-se o vetor de coeficientes de custo da função objetivo e vai-se preenchendo os campos correspondentes do vetor armazenador para cada coeficiente de custo analisado.

Neste exemplo o vetor armazenador assumirá os seguintes valores:

ARMAZENADOR(0,1) = 4

ARMAZENADOR(13,1) = 1

ARMAZENADOR(21,1) = 5

ARMAZENADOR(22,1) = 9

ARMAZENADOR(54,1) = 8

ARMAZENADOR(66,1) = 2

ARMAZENADOR(66,2) = 3

ARMAZENADOR(79,1) = 7

ARMAZENADOR(80,1) = 10

ARMAZENADOR(84,1) = 8, os elementos restantes do vetor armazenador continuam com valor zero.

Como se vê, neste instante, tem-se que todos os coeficientes de custo, bem como o número das variáveis correspondentes a eles, estão armazenados no vetor ARMAZENADOR.

É necessário que o ordenamento seja feito em ordem decrescente. Para que isto seja possível basta que se percorra o vetor ARMAZENADOR em ordem decrescente partindo de um valor igual a RANGE - 1 (neste exemplo, corresponde a 84) até zero inclusive, tendo-se o cuidado de eliminar as componentes que tiverem VAR = 0 (pois estes campos do vetor ARMAZENADOR, não correspondem a nenhuma variável do problema em questão).

A medida que é percorrido o vetor ARMAZENADOR, a cada vez que se encontra um valor de VAR diferente de zero, coloca-se este valor em um outro vetor que, ao fim da passagem pelo vetor armazenador, estará ordenado e separado pelo valor das variáveis (0 ou 1).

O vetor ordenado poderá ser modelado da seguinte forma:

ORD_COL (1 : NUM_INT ; 0 : 1) = VAR

Coef. 1 Coef.2 Coef.3

onde:

Coef. 1 : equivale ao número de ordem do vetor; portanto o valor de custo máximo tem valor de Coef. 1 igual a 1, o segundo maior valor tem o Coef. 2 igual a 2 e assim por diante. No momento em que o Coef. 1 é incrementado, adiciona-se, também, uma unidade a um contador do número de variáveis que estão em 0 ou em 1 (CONTO1(0:1)), portanto, ao final da passagem, CONTO1(0) conterá o número de variáveis que estão com valor 0 e CONTO1(1) conterá o número das variáveis que estão com valor 1 (neste caso, CONTO1(0) = 4 e CONTO1(1) = 6).

Coef. 2 : assume o valor 0 ou 1 dependendo do valor que a variável correspondente ao coeficiente de custo em questão tem na solução dada.

Coef. 3 : corresponde ao número da variável associada ao custo em questão.

Inicialmente todos os elementos do vetor ORD_COL assumem o valor 0.

Após percorrer o vetor ARMAZENADOR da maneira descrita anteriormente, e ter armazenado os valores de VAR diferentes de 0

no vetor ORD_COL, tem-se todas as variáveis ordenadas em ordem decrescente em relação ao seu coeficiente de custo e, simultaneamente, separadas em relação ao valor que possuiam na solução dada (0 ou 1).

Neste caso o vetor ORD_COL teria a seguinte forma:

ORD_COL(1,0) = 10
ORD_COL(2,0) = 7
ORD_COL(3,0) = 2
ORD_COL(4,0) = 6

ORD_COL(1,1) = 8
ORD_COL(2,1) = 3
ORD_COL(3,1) = 9
ORD_COL(4,1) = 5
ORD_COL(5,1) = 1
ORD_COL(6,1) = 4

e os contadores de número de variáveis teriam os seguintes valores: CONTO1(0) = 4 e CONTO1(1) = 6.

Com isto, está concluída a descrição desta adaptação do Método Radix para ordenação de números inteiros. A seguir discutem-se as alterações feitas no sentido de fazer a busca dos movimentos simples, duplos e triplos mais rápida e racional, levando-se em conta os fundamentos da busca tabu.

3.3. BUSCA DE UM COMPLEMENTO SIMPLES NA FASE DE MELHORAMENTO

O complemento simples na fase de melhoramento busca uma

única variável que passe de um de seus limites para o outro, trazendo uma melhora no valor da solução atual, enquanto mantém a factibilidade desta solução.

Sabendo que $c_j \geq 0$ e que a eficiência da busca tabu está relacionada com a procura do complemento que traga o maior incremento possível no valor da função objetivo (no contexto de maximização da função objetivo), só se apresentam como candidatas a passar de 0 para 1 as variáveis que não tenham coeficientes de custo nulos.

Portanto, na busca do melhor complemento simples possível, basta que se percorra a lista ORD_COL, das variáveis que estão em 0 em ordem crescente (neste caso de 1 até CONTO1(0) = 4) até que uma das seguintes condições seja satisfeita:

- É encontrado um complemento simples factível, ou

- É encontrada uma variável com valor de coeficiente de custo nulo (a partir deste instante, não vale a pena continuar a busca, pois todas as variáveis seguintes tem valor de coeficiente de custo também nulo; portanto, não trazem melhora ao valor da solução), ou

- É encontrado o fim da lista.

Neste exemplo, a ordem em que os complementos candidatos são analisados é a seguinte:

1) ORD_COL(1,0) = 10 ; c_{10} = 80

2) ORD_COL(2,0) = 7 ; c_{7} = 79

3) ORD_COL(1,0) = 2 ; c_{2} = 66

4) ORD_COL(1,0) = 6 ; c_{6} = 54

Note que o primeiro complemento simples encontrado é o que maior incremento traz ao valor da solução atual. Caso não haja nenhum complemento simples factível, passa-se a analisar a possibilidade de se realizar um complemento duplo na fase de melhoramento.

3.4. BUSCA DE UM COMPLEMENTO DUPLO NA FASE DE MELHORAMENTO

O complemento duplo na fase de melhoramento busca um par de variáveis cuja complementação produza uma melhora na solução atual, mantendo a factibilidade desta solução.

Na análise de um complemento duplo, têm-se duas situações a considerar: a primeira é aquela onde uma das variáveis passa de 0 para 1 e a outra de 1 para 0 e a segunda situação é aquela onde as duas variáveis passam de 0 para 1. Perceba que a situação onde as duas variáveis passam de 1 para 0

não é analisada, pois tem-se que todo $c_j \geq 0$ e, portanto, este tipo de complemento não traz melhora alguma ao objetivo.

Seguindo a orientação da busca tabu, organizou-se uma estratégia de busca de modo que sejam analisados primeiramente aqueles complementos que provocam os maiores incrementos no valor da solução.

Para que isto seja possível, inicialmente analisa-se o caso onde uma variável passa de 0 para 1 e a outra de 1 para 0 e, para achar o complemento duplo de maior incremento, percorre-se a lista ORD_COL das variáveis que estão em 1 em ordem decrescente (neste exemplo, de 6 até 1) e para cada valor da lista ORD_COL das variáveis que estão em 1, percorre-se a lista ORD_COL das variáveis que estão em 0 em ordem crescente (neste exemplo, de 1 até 4) e interrompe-se a busca numa das seguintes situações:

- É encontrado um complemento duplo factível, ou

- É encontrado o fim das listas.

Neste exemplo a busca seria dirigida seguindo esta seqüência:

1) ORD_COL(6,1) = 4 e ORD_COL(1,0) = 10 ; $c_{10} - c_4 = 80$

2) ORD_COL(6,1) = 4 e ORD_COL(2,0) = 7 ; $c_7 - c_4 = 79$

3) ORD_COL(6,1) = 4 e ORD_COL(3,0) = 2 ; $c_2 - c_4 = 66$

4) ORD_COL(6,1) = 4 e ORD_COL(4,0) = 6 ; $c_6 - c_4 = 54$

5) ORD_COL(5,1) = 1 e ORD_COL(1,0) = 10 ; $c_{10} - c_1 = 67$

6) ORD_COL(5,1) = 1 e ORD_COL(2,0) = 7 ; $c_7 - c_1 = 66$

⋮ ⋮ ⋮

e assim por diante até que o fim das listas seja encontrado ou que um complemento duplo factível seja encontrado.

Note que os primeiros complementos analisados são aqueles com maior probabilidade de dar um incremento maior no valor da solução atual. Caso não seja encontrado nenhum complemento duplo factível na busca acima, deve-se partir para a segunda situação.

A segunda situação é aquela onde as duas variáveis passam de 0 para 1. Analogamente, para se analisar primeiro os maiores incrementos, percorre-se a lista ORD_COL das variáveis que estão em 0 em ordem crescente partindo de 1 até CONTO1(0) - 1 (neste exemplo, de 1 até 3) e para cada valor da lista ORD_COL das variáveis que estão em 0, percorre-se esta mesma lista a partir deste valor em ordem crescente até CONTO1(0).

Neste exemplo tem-se a seguinte ordem de busca:

1) ORD_COL(1,0) = 10 e ORD_COL(2,0) = 7 ; $c_{10} + c_7 = 159$

2) ORD_COL(1,0) = 10 e ORD_COL(3,0) = 2 ; $c_{10} + c_2 = 146$

3) ORD_COL(1,0) = 10 e ORD_COL(4,0) = 6 ; $c_{10} + c_6 = 134$

4) ORD_COL(2,0) = 7 e ORD_COL(3,0) = 2 ; $c_7 + c_2 = 145$

5) ORD_COL(2,0) = 7 e ORD_COL(4,0) = 6 ; $c_7 + c_6 = 133$

6) ORD_COL(3,0) = 2 e ORD_COL(4,0) = 6 ; $c_2 + c_6 = 120$

Aqui também realiza-se o primeiro complemento duplo factível encontrado ou percorre-se a lista até o fim.

Vale observar que no caso de problemas onde todos os coeficientes são restritos a valores não negativos, a análise desta segunda situação se restringe ao caso onde uma das variáveis envolvidas no complemento tem valor de coeficiente de custo nulo, pois é o único caso não coberto pela realização de dois complementos simples, que são preferenciais em relação ao duplo.

Como o complemento duplo ainda é uma operação de custo computacional relativamente baixo, optou-se por continuar com uma busca exaustiva, mas de uma maneira bem mais inteligente, procurando analisar primeiro os complementos que levem a um maior incremento. Com isto espera-se ganhar tempo e qualidade na busca.

Em relação ao complemento triplo, optou-se por executá-lo somente depois de se realizar o primeiro movimento reverso, pois o número de possibilidades torna o complemento triplo muito custoso computacionalmente.

Portanto, a análise do complemento triplo só será feita se não for encontrado nenhum complemento duplo factível e já

tiver sido realizado pelo menos um movimento reverso. A forma de escolha do complemento triplo a realizar na fase de melhoramento é o que é relatado em seguida.

3.5. BUSCA DE UM COMPLEMENTO TRIPLO NA FASE DE MELHORAMENTO

Na fase de melhoramento, o complemento triplo busca por um trio de variáveis cuja complementação produza uma melhora na solução atual, mantendo a factibilidade desta solução.

Na busca de um complemento triplo tem-se três situações a analisar: a primeira é aquela onde duas variáveis passam de 0 para 1 e a outra de 1 para 0; a segunda situação é aquela onde duas variáveis passam de 1 para 0 e a outra de 0 para 1 e por último tem-se a situação onde as três variáveis passam de 0 para 1.

Note que a situação onde três variáveis passam de 1 para 0 não precisa ser analisada, pois não traz nenhuma melhora no valor da solução corrente.

Como já mencionado, a busca exaustiva de um complemento triplo tem um custo computacional muito elevado. Assim, ao invés de se fazer uma busca exaustiva do complemento triplo, optou-se por uma busca em determinadas regiões que pareçam ser mais

promissoras no sentido de conterem complementos que provoquem bons acréscimos na função objetivo.

Esta alternativa foi sugerida por GLOVER [15], que diz que, se o custo de cálculo é elevado, pode-se fazer uma redução do espaço amostral diminuindo a região de busca e concentrando a atenção em regiões que pareçam ser mais promissoras.

Seguindo este espírito, adaptou-se o algoritmo de modo que a busca por um complemento triplo deixasse de ser exaustiva e passase a ser mais racional, sem entretanto, perder a qualidade da solução encontrada.

A primeira situação é aquela onde duas variáveis passam de 0 para 1 e a outra de 1 para 0. Para analisar este caso, percorre-se a lista ORD_COL das variáveis que estão em 0 em ordem crescente de 1 até CONTO1(0) - 1 (neste exemplo, de 1 até 3) fixando-se a primeira variável na lista e a sua imediatamente posterior; e, para cada par fixado, percorre-se a lista ORD_COL das variáveis que estão em 1 em ordem decrescente partindo de CONTO1(1) até 1 (neste exemplo, de 6 até 1). A busca se interrompe numa das seguintes situações:

- É encontrado um complemento triplo factível ou,

- É encontrado o fim das listas.

Neste exemplo tem-se a seguinte ordem de busca:

1) ORD_COL(1,0)=10, ORD_COL(2,0)=7 e ORD_COL(6,1)=4; $c_{10}+c_7-c_4=159$

2) ORD_COL(1,0)=10, ORD_COL(2,0)=7 e ORD_COL(5,1)=1; $c_{10}+c_7-c_1=146$

3) ORD_COL(1,0)=10, ORD_COL(2,0)=7 e ORD_COL(4,1)=5; $c_{10}+c_7-c_5=138$

4) ORD_COL(1,0)=10, ORD_COL(2,0)=7 e ORD_COL(3,1)=9; $c_{10}+c_7-c_9=137$

5) ORD_COL(1,0)=10, ORD_COL(2,0)=7 e ORD_COL(2,1)=3; $c_{10}+c_7-c_3=93$

6) ORD_COL(1,0)=10, ORD_COL(2,0)=7 e ORD_COL(1,1)=8; $c_{10}+c_7-c_8=75$

7) ORD_COL(2,0)=7, ORD_COL(3,0)=2 e ORD_COL(6,1)=4; $c_7+c_2-c_4=145$

⋮ ⋮ ⋮ ⋮

e assim por diante até que um complemento triplo factível seja encontrado ou as listas cheguem ao final.

Pode-se notar que o espaço de busca ficou reduzido, entretanto, foram analisados desde um complemento com grande incremento no valor da solução até um pequeno incremento, o que permite concluir que este novo espaço de busca apesar de reduzido abrange um largo espectro de soluções.

A segunda situação a ser analisada é aquela em que duas variáveis passam de 1 para 0 e a outra variável passa de 0 para 1. Seguindo o mesmo raciocínio acima apresentado, faz-se a busca do complemento triplo percorrendo a lista ORD_COL das variáveis que estão em 1 em ordem decrescente, partindo de CONTO1(1) até 2 (neste exemplo, de 6 até 2) fixando a primeira variável da lista e a sua imediatamente anterior; e, para cada par fixado, percorre-se a lista ORD_COL das variáveis que estão em 0 em ordem

crescente de 1 até CONTO1(O) (neste exemplo, de 1 até 4) até que uma das situações se verifique:

- É encontrado um complemento triplo factível, ou

- É encontrado o fim das listas.

Neste exemplo tem-se a seguinte ordem de busca:

1) ORD_COL(6,1)=4, ORD_COL(5,1)=1 e ORD_COL(1,O)=10; $c_{10}-c_4-c_1=67$

2) ORD_COL(6,1)=4, ORD_COL(5,1)=1 e ORD_COL(2,O)=7 ; $c_7-c_4-c_1=66$

3) ORD_COL(6,1)=4, ORD_COL(5,1)=1 e ORD_COL(3,O)=2 ; $c_2-c_4-c_1=53$

4) ORD_COL(6,1)=4, ORD_COL(5,1)=1 e ORD_COL(4,O)=6 ; $c_6-c_4-c_1=41$

5) ORD_COL(5,1)=1, ORD_COL(4,1)=5 e ORD_COL(1,O)=10; $c_{10}-c_1-c_5=46$

⋮ ⋮ ⋮ ⋮

e assim sucessivamente até que um complemento triplo factível seja encontrado ou as listas cheguem ao fim.

A terceira situação a ser analisada é aquela onde as três variáveis passam de 0 para 1. Analogamente, procura-se percorrer a lista ORD_COL das variáveis que estão em 0 em ordem crescente de 1 até CONTO1(O) - 2 (neste exemplo, de 1 até 2) e trata-se com as três variáveis, a partir da primeira em questão na lista, mais as suas duas subseqüentes, até que uma das situações se verifique:

- É encontrado um complemento triplo factível, ou

- É encontrado o fim da lista.

Neste exemplo tem-se a seguinte ordem de busca:

1)ORD_COL(1,0)=10, ORD_COL(2,0)=7 e ORD_COL(3,0)=2; $c_{10}+c_7+c_2=225$

2)ORD_COL(2,0)=7 , ORD_COL(3,0)=2 e ORD_COL(4,0)=6; $c_7+c_2+c_6=211$

A mesma observação feita anteriormente para os complementos duplos pode ser aqui reiterada, ou seja, se todos os coeficientes do problema forem não negativos, só interessam os complementos triplos, nesta terceira situação, que envolvam pelo menos uma variável com coeficiente de custo nulo.

Note que o espaço de busca do complemento triplo fica bastante reduzido, o que coloca seu custo computacional numa posição de competição com os custos dos complementos simples e duplos. Com isto é possível almejar soluções de melhor qualidade com esforço computacional relativamente pequeno.

3.6. BUSCA DE UM COMPLEMENTO SIMPLES NA FASE TABU

Na fase tabu a busca por um complemento simples resume-se em escolheruma variável que passe de um de seus limites para o outro, resultando numa piora do valor da solução atual, enquanto mantém a factibilidade desta solução.

Sabendo que todo $c_j \geq 0$ e que a filosofia que norteia a busca tabu estabelece que quando for necessário fazer um movimento que piore a solução atual, que este movimento traga o menor decréscimo possível na solução, vê-se que nesta fase só interessará a analise das variáveis que puderem passar de 1 para 0 e ainda com preferência pelas variáveis que tenham coeficiente de custo nulo, pois o complemento delas não altera o valor da solução corrente.

A partir de agora, chamar-se-á complemento simples na fase tabu de *reverso simples*; portanto, na busca do melhor reverso simples possível, basta que se percorra a lista ORD_COL, das variáveis que estão em 1 em ordem decrescente (neste caso de CONTO1(1) = 6 até 1) até que uma das seguintes condições seja satisfeita:

- É encontrado um reverso simples factível, ou

- É encontrado o fim da lista.

Neste exemplo, a ordem em que os complementos candidatos são analisados é a seguinte:

1) ORD_COL(6,1) = 4 ; $-c_4$ = 0

2) ORD_COL(5,1) = 1 ; $-c_1$ = -13

3) ORD_COL(4,1) = 5 ; $-c_5$ = -21

4) ORD_COL(3,1) = 9 ; $-c_9$ = -22

5) ORD_COL(2,1) = 3 ; $-c_3$ = -66

6) ORD_COL(1,1) = 8 ; $-c_8$ = -84

Note que dentre os reversos simples, o primeiro encontrado é o que menor decréscimo traz ao valor da solução atual.

Caso não haja nenhum reverso simples factível, passa-se a analisar a possibilidade de realizar um complemento duplo na fase tabu, que passará a ser chamado *reverso duplo*.

3.7. BUSCA DE UM COMPLEMENTO DUPLO NA FASE TABU

O reverso duplo busca um par de variáveis cuja complementação produza uma piora na solução atual, mantendo a factibilidade da mesma.

Na análise de um reverso duplo, tém-se duas situações a considerar: a primeira é aquela onde uma das variáveis passa de 0 para 1 e a outra de 1 para 0 e a segunda situação é aquela onde as duas variáveis passam de 1 para 0. A situação onde as duas variáveis passam de 0 para 1 não é analisada, pois como todo $c_j \geq 0$, este tipo de complemento não é do tipo reverso.

Seguindo a orientação de busca tabu, organizou-se uma

estratégia de busca que procura, primeiramente, os complementos que provoquem menores pioras no valor da solução corrente.

A estratégia elaborada analisa, inicialmente, o caso onde uma variável passa de 0 para 1 e a outra de 1 para 0 e, para que se tenha os menores decrementos em primeiro plano, percorre-se a lista ORD_COL das variáveis que estão em 1 em ordem decrescente (neste exemplo de CONTO1(1) = 6 até 1) e para cada valor da lista ORD_COL das variáveis que estão em 1, percorre-se a lista ORD_COL das variáveis que estão em 0 em ordem decrescente (neste exemplo de CONTO1(0) = 4 até 1) e interrompe-se a busca numa das seguintes situações:

- É encontrado um reverso duplo factível, ou

- É encontrado o fim das listas.

Neste exemplo a busca é dirigida seguindo esta seqüência:

1) ORD_COL(6,1) = 4 e ORD_COL(4,0) = 6 ; $-c_4 + c_6 = 54$

2) ORD_COL(6,1) = 4 e ORD_COL(3,0) = 2 ; $-c_4 + c_2 = 66$

3) ORD_COL(6,1) = 4 e ORD_COL(2,0) = 7 ; $-c_4 + c_7 = 79$

4) ORD_COL(6,1) = 4 e ORD_COL(1,0) = 10 ; $-c_4 + c_{10} = 80$

5) ORD_COL(5,1) = 1 e ORD_COL(4,0) = 6 ; $-c_1 + c_6 = 41$

6) ORD_COL(5,1) = 1 e ORD_COL(3,0) = 2 ; $-c_1 + c_2 = 53$

⋮ ⋮ ⋮

e assim por diante até que o fim das listas seja encontrado ou um reverso duplo factível seja encontrado.

Note que os primeiros complementos analisados são aqueles com maior probabilidade de dar um menor decréscimo no valor da solução atual. Caso não seja encontrado nenhum reverso duplo factível na busca acima, deve-se partir para a segunda situação.

A segunda situação é aquela onde as duas variáveis passam de 1 para 0. Para se obterem os menores decrementos, percorre-se a lista ORD_COL das variáveis que estão em 1 em ordem decrescente partindo de CONTO1(1) até 2 (neste exemplo, de 6 até 2) e para cada valor da lista ORD_COL das variáveis que estão em 1, percorre-se o restante desta mesma lista, a partir do valor corrente, em ordem crescente até CONTO1(1).

Neste exemplo tem-se a seguinte ordem de busca:

1) ORD_COL(6,1) = 4 e ORD_COL(5,1) = 1 ; $-c_4 - c_1 = -13$

2) ORD_COL(6,1) = 4 e ORD_COL(4,1) = 5 ; $-c_4 - c_5 = -21$

3) ORD_COL(6,1) = 4 e ORD_COL(3,1) = 9 ; $-c_4 - c_9 = -22$

4) ORD_COL(6,1) = 4 e ORD_COL(2,1) = 3 ; $-c_4 - c_3 = -66$

5) ORD_COL(6,1) = 4 e ORD_COL(1,1) = 8 ; $-c_4 - c_8 = -84$

6) ORD_COL(5,1) = 1 e ORD_COL(4,1) = 5 ; $-c_1 - c_5 = -34$

7) ORD_COL(5,1) = 1 e ORD_COL(3,1) = 9 ; $-c_1 - c_9 = -43$

⋮ ⋮ ⋮

e realiza-se o primeiro reverso duplo factível que se encontrar ou percorre-se a lista até o fim.

Como comentado anteriormente, o reverso duplo é uma operação de custo computacional relativamente baixo. Portanto continua-se fazendo uma busca exaustiva, mas de uma maneira mais racional, procurando colocar em primeiro plano aqueles reversos que causem um menor decremento na solução corrente, e consequentemente, melhorando a qualidade da busca.

Quando não se encontra nenhum reverso duplo factível, tenta-se ainda, a realização de um complemento triplo na fase tabu, que será chamado de *reverso triplo*.

3.8. BUSCA DE UM COMPLEMENTO TRIPLO NA FASE TABU

O reverso triplo busca por um trio de variáveis cuja complementação produza uma piora na solução atual, mantendo a factibilidade desta solução.

Aqui segue-se a mesma filosofia do complemento triplo na fase de melhoramento, ou seja, concentra-se a busca em regiões que pareçam ser mais promissoras, diminuindo assim, o custo computacional gasto na realização deste tipo de movimento.

É interessante salientar novamente que o algoritmo sai

da fase de melhoramento quando não é possível achar um movimento que melhore o valor da solução corrente. Esta análise é feita exaustivamente para os complementos simples e duplos, mas para os triplos só é analisado um subconjunto dos complementos possíveis. Esta redução do espaço de busca é compensada pela criação de um mecanismo inteligente para a busca de bons complementos triplos, alicerçado na ordenação das variáveis da solução corrente. Com isso espera-se uma substancial diminuição do tempo de busca, sem perda de qualidade nos movimentos triplos realizados.

Assim, quando o algoritmo passa para a fase tabu, é certo que os movimentos simples e duplos a serem analisados ou provocam uma piora, ou são infactíveis, ou são proibidos. Entretanto, para os movimentos triplos é possível, mas pouco provável, que haja movimentos factíveis, não proibidos e de *melhora* da função. Porém, como parte da estratégia de poupar tempo na busca por complementos triplos, optou-se por procurar somente aqueles movimentos que provoquem *piora* na função objetivo.

Isto se justifica devido a inteligência propiciada pela busca ordenada, que reduz a probabilidade de haver bons movimentos triplos de melhora na fase tabu. Assim, mesmo parecendo absurdo desprezar um movimento triplo que provoque melhora da função objetivo em favor de um outro que provoque piora, optou-se por esse procedimento confiando na baixa probabilidade dele ocorrer. Por outro lado, ganha-se tempo por

não ter que verificar a factibilidade de todos os movimentos, mas somente daqueles de piora.

Na busca de um reverso triplo têm-se três situações a analisar: a primeira é aquela onde duas variáveis passam de 0 para 1 e a outra de 1 para 0; a segunda situação é aquela onde duas variáveis passam de 1 para 0 e a outra de 0 para 1 e, por último, tem-se a situação onde as três variáveis passam de 1 para 0.

Neste caso não se analisa a situação onde as três variáveis passariam de 0 para 1, pois este caso só traria melhoras no valor da solução corrente e não é o que se busca no momento.

A metodologia não exaustiva usada aqui será descrita a seguir para cada uma das três situações anteriormente mencionadas.

Na primeira situação percorre-se a lista ORD_COL das variáveis que estão em 0 em ordem decrescente de CONTO1(0) até 2 (neste exemplo, de 4 até 2) fixando a primeira variável em questão na lista e a sua imediatamente anterior e, para cada par fixado, a lista ORD_COL das variáveis que estão em 1 é percorrida em ordem decrescente partindo de CONTO1(1) até 1 (neste exemplo, de 6 até 1). A busca termina numa das seguintes situações:

- É encontrado um reverso triplo factível, ou

- É encontrado o fim das listas.

Neste exemplo teremos a seguinte ordem de buscas:

1) ORD_COLC(4,0)=6, ORD_COLC(3,0)=2 e ORD_COLC(6,1)=4 ; $c_6+c_2=c_4$?

2) ORD_COLC(4,0)=6, ORD_COLC(3,0)=2 e ORD_COLC(5,1)=1 ; $c_6+c_2=c_1$?

3) ORD_COLC(4,0)=6, ORD_COLC(3,0)=2 e ORD_COLC(4,1)=5 ; $c_6+c_2=c_5$?

4) ORD_COLC(4,0)=6, ORD_COLC(3,0)=2 e ORD_COLC(3,1)=3 ; $c_6+c_2=c_3$?

5) ORD_COLC(4,0)=6, ORD_COLC(3,0)=2 e ORD_COLC(2,1)=8 ; $c_6+c_2=c_8$?

6) ORD_COLC(4,0)=6, ORD_COLC(3,0)=2 e ORD_COLC(1,1)=8 ; $c_6+c_2=c_8$?

7) ORD_COLC(3,0)=2, ORD_COLC(2,0)=7 e ORD_COLC(6,1)=4 ; $c_2+c_7=c_4$?

⋮ ⋮ ⋮ ⋮

e assim por diante até que um reverso triplo factível seja encontrado ou as listas cheguem ao final.

Vê-se que neste exemplo, nenhum reverso triplo havia analisado, pois todos os possíveis candidatos trariam melhoria na solução atual. Porém, com alta probabilidade, eles seriam ou não-factíveis ou proibidos e, por isso, descartados anteriormente na fase de melhoramento.

[illegible] decrescente partindo de CONTOI(1) até 2 (neste exemplo, de 6 até 2) fixando a primeira variável em questão da lista e a sua

imediatamente anterior. Para cada par fixado, percorre-se a lista ORD_COL das variáveis que estão em 0 em ordem decrescente de CONTO1(0) até 1(neste exemplo, de 4 até 1), até que uma das situações se verifique:

- É encontrado um complemento triplo factível, ou

- É encontrado o fim das listas.

Neste exemplo tem-se a seguinte ordem de busca:

1) ORD_COL(6,1)=4, ORD_COL(5,1)=1 e ORD_COL(4,0)=6 ; $-c_4-c_1+c_6$ =41

2) ORD_COL(6,1)=4, ORD_COL(5,1)=1 e ORD_COL(3,0)=2 ; $-c_4-c_1+c_2$ =53

3) ORD_COL(6,1)=4, ORD_COL(5,1)=1 e ORD_COL(2,0)=7 ; $-c_4-c_1+c_7$ =66

4) ORD_COL(6,1)=4, ORD_COL(5,1)=1 e ORD_COL(1,0)=10; $-c_4-c_1+c_{10}$ =67

5) ORD_COL(5,1)=1, ORD_COL(4,1)=5 e ORD_COL(4,0)=6 ; $-c_1-c_5+c_6$ =20

6) ORD_COL(5,1)=1, ORD_COL(4,1)=5 e ORD_COL(3,0)=2 ; $-c_1-c_5+c_2$ =32

⋮ ⋮ ⋮ ⋮

e assim por diante até que um reverso triplo factível seja encontrado ou as listas cheguem ao fim.

A terceira situação a ser analisada é aquela onde as três variáveis passam de 1 para 0. Assim, percorre-se a lista ORD_COL das variáveis que estão em 1 em ordem decrescente de CONTO1(1) até 3 (neste exemplo, de 6 até 3) e percorrendo as três variáveis, à partir da primeira em questão na lista, mais as duas imediatamente antecessoras, até que uma das situações se

verificação:

- É encontrado o [illegible] final, ou

- É encontrado o fim da lista.

Neste exemplo tem-se a seguinte ordem de busca:

1)ORD_COLC(5,1)=4, ORD_COLC(5,1)=1 e ORD_COLC(4,1)=5; $c_4 c_1 c_5$ = 4

2)ORD_COLC(5,1)=1, ORD_COLC(4,1)=5 e ORD_COLC(3,1)=3; $c_1 c_5 c_3$ = 24

3)ORD_COLC(4,1)=5, ORD_COLC(3,1)=3 e ORD_COLC(2,1)=3; $c_5 c_3 c_3$ =100

4)ORD_COLC(3,1)=3, ORD_COLC(2,1)=3 e ORD_COLC(1,1)=3; $c_3 c_3 c_3$ = 12

Analogamente à fase de melhoramento, o reverso triplo teve seu espaço de busca bastante reduzido, o que diminui consideravelmente o esforço computacional dispendido na busca do mesmo. Como esta redução foi feita de uma maneira racional, isto nos leva a supor que serão encontradas soluções de melhor qualidade de uma forma eficiente.

3.9. ALGORITMO INCLUINDO BUSCA TABU COM MELHORIAS

O algoritmo heurístico [illegible] busca tabu com as [illegible] discutidas.

A fase de busca do algoritmo não sofreu mudanças em relação a versão anterior, porém na fase de melhoramento foram acrescentados dois passos e houve mudanças em alguns passos devido à ordenação. Na fase tabu também houve mudanças.

Na fase de melhoramento foram acrescentados os Passos 2 e 5 e o Passo 3 sofreu mudanças.

O Passo 2 se refere à ordenação feita nas variáveis. Esta ordenação é feita cada vez que algum complemento é realizado, pois a solução foi mudada.

Em relação ao Passo 5 percebe-se que o algoritmo com ordenação fica bem mais poderoso e assim, optou-se por não realizar complementos triplos na fase de melhoramento, só liberando a realização deste tipo de movimento após ter sido realizado algum movimento reverso. Isto porque supõe-se que a atual solução se encontra numa região mais próxima do ótimo, justificando a realização de um movimento triplo, que é custoso computacionalmente.

No Passo 3 não se busca mais a variável que produz o maior incremento na função objetivo, e sim a primeira variável encontrada [illegible] não [illegible] função objetivo.

Na fase tabu a mudança ocorreu no Passo 2 do algoritmo,

onde não se busca mais a variável que produz o maior decréscimo na função objetivo, e sim a primeira variável encontrada, pois, com a ordenação, esta primeira variável é a que traz o maior decréscimo na função objetivo.

FASE DE BUSCA

[Passo 1] Resolva o (PL). Se a solução for inteira PARE, ela é ótima.
Caso contrário, vá para o Passo 2.

[Passo 2] Busque um pivot do tipo 1, se não existir vá para o Passo 3.
Caso contrário, execute o pivot do tipo 1 que resulte no maior valor da função objetivo e vá para o Passo 4.

[Passo 3] Busque um pivot do tipo 2, se não existir vá para o Passo 5.
Caso contrário, execute o primeiro pivot do tipo 2 encontrado e vá para o Passo 4.

[Passo 4] Examine a solução básica atual, se ela é inteira vá para a FASE DE APERFEIÇOAMENTO.
Caso contrário, vá para o Passo 2.

[Passo 5] Verifique se arredondando ou truncando a solução [illegible] atual, produz-se uma solução 0-1 factível. Se sim, vá para a FASE DE MELHORAMENTO.
Caso contrário, vá para o Passo 6.

[Passo 6] Execute um pivot do tipo 3 que minimize o valor da medida de infactibilidade e vá para o Passo 7.

[Passo 7] Busque uma única variável 0-1, não básica, cuja complementação reduz o valor da medida de infactibilidade. Se não existir nenhuma, vá para o Passo 9.
Caso contrário, complemente a variável que produz a maior redução na medida de infactibilidade e vá para o Passo 8.

[Passo 8] Se a solução atual é infactível, vá para o Passo 7.
Se ela é factível, verifique se arredondando ou truncando a solução atual produz-se uma solução 0-1 factível. Se sim, vá para a FASE DE MELHORAMENTO. Caso contrário, vá para o Passo 9.

[Passo 9] Busque um par de variáveis 0-1, não básicas, cuja complementação reduz o valor atual da medida de infactibilidade. Se não houver nenhum, PARE, o procedimento heurístico FALHOU (não encontrou nenhuma solução 0-1 factível).
Caso contrário, complemente o primeiro par identificado e vá para o Passo 8.

Fase de Melhoramento

[Passo 1] Fixe todas as variáveis 0-1 que possam ser fixadas e vá para o Passo 2.

[Passo 2] Faça um ordenamento conveniente nas variáveis 0-1 candidatas a ser complementadas e vá para o Passo 3.

[Passo 3] Busque uma única variável 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um incremento na solução 0-1. Se não existir vá para o Passo 4.
Caso contrário, complemente a primeira variável encontrada, atualize o movimento proibido e vá para o Passo 1.

Passo 4 Busque um par de variáveis 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um incremento na solução 0-1. Se não existir vá para o Passo 5.

Caso contrário, complemente o primeiro par candidato, proiba o movimento direto e o seu reverso e vá para o Passo 1.

Passo 5 Verifique se já foi realizado algum movimento reverso; se sim vá para o Passo 6.

Caso contrário vá para a FASE TABU.

Passo 6 Busque um trio de variáveis 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um incremento na solução 0-1. Se não existir vá para a FASE TABU.

Caso contrário, complemente o primeiro trio candidato, proiba o movimento direto e o seu reverso e vá para o Passo 1.

FASE TABU

Passo 1 Teste os critérios de parada do algoritmo. Se algum for satisfeito, exiba a melhor solução 0-1 encontrada até o momento e PARE.

Caso contrário, vá para o Passo 2.

[Passo 2] Busque uma única variável 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um decremento na solução 0-1. Se não existir, vá para o Passo 3.

Caso contrário, complemente a primeira variável encontrada, proiba o movimento direto e o seu reverso e vá para a FASE DE MELHORAMENTO.

[Passo 3] Busque um par de variáveis 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um decremento na solução 0-1. Se não existir, vá para o Passo 4.

Caso contrário, complemente o primeiro par candidato, proiba o movimento direto e o seu reverso e vá para a FASE DE MELHORAMENTO.

[Passo 4] Busque um trio de variáveis 0-1 cuja complementação é possível (Movimento não Proibido) e que produz um decremento na solução 0-1. Se não existir vá para o Passo 1.

Caso contrário, complemente o primeiro trio candidato, proiba o movimento direto e o seu reverso e vá para a FASE DE MELHORAMENTO.

CAPÍTULO 4

RESULTADOS COMPUTACIONAIS E ANÁLISE DE SENSIBILIDADE DO ALGORITMO INCLUINDO BUSCA TABU

4.1. RESULTADOS COMPUTACIONAIS

Novos testes computacionais foram realizados para avaliar as melhorias introduzidas no algoritmo.

Para estes testes, foi usado o mesmo conjunto de problemas anteriormente avaliado.

Os resultados podem ser vistos na tabela 4.1.

Nome	n	m	Zot	(Zot - Zpc)/Zot (x 100%)	(Zot - Ztb)/Zot (x 100%)	P	E	B	N	Tt (seg.)	Tpc (seg.)	Ttb (seg.)	Nº
PETER3	15	10	4015	18,431	0,000	3	52	12	23	16,2	3,0	3,4	
PETER4	20	10	6120	0,490	0,327	10	53	13	31	71,3	0,8	4,3	
PETER5	28	10	12400	4,113	0,000	15	51	1	15	109,0	0,5		
PETER6	39	5	10618	2,515	0,251	6	52	12	24	210,7	0,6	3,0	
PETER7	50	5	16537	0,230	0,230	12	52	15	2	237,1	1,8	4,9	10,
CEM53	100	5	5472,51	0,033	0,004	43	50	19	6	106,1	7,3	8,2	97,
CEM54	100	5	3019,17	0,166	0,136	23	51	9	7	52,0	5,6	6,	
DUZ51	200	5	6755,60	0,145	0,057	155	52	18	7	1.2.1	17,1	18,	
DUZ53	200	5	6247,44	0,199	0,151	104	50	21	15	420,0	7,2	9,3	84,
CEM103	100	10	4529,71	0,582	0,512	33	51	17	34	208,8	6,2	9,4	
CEM108	100	10	3112,64	0,720	0,694	24	50	17	19	271,9	6,2	4,4	
DUZ101	200	10	5998,90	0,932	0,932	0	52	13	15	2907,1	3,0	11,0	65,
DUZ104	200	10	6602,66	0,207	0,192	77	52	16	10	560,4	14,8	2,6	
P2	50	35	167,443	37,820	27,139	1	50	14	98	100,3	27,1	10,2	
P3	100	30	250,087	12,277	5,744	0	50	13	33	404,2	14,6	10,2	15,
P4	200	30	1513,14	4,834	3,842	2	51	11	34	2251,6	11,9	10,9	37,

TABELA 4.1 - DESEMPENHO DO ALGORITMO TABU COM MELHORIAS

onde:

- n - número de variáveis 0-1 do problema;
- m - número de restrições do problema;
- Zot - nas séries PETER, corresponde ao valor da solução ótima do problema 0-1,
  nas séries CEM, DUZ e P, corresponde ao valor da solução ótima do problema linear relaxado;
- Zpc - valor de solução 0-1 encontrado pela heurística pivot e complemento (Balas e Martin);
- Ztb - valor de solução 0-1 encontrado pela aplicação da busca tabu;
- P[illegible] - [illegible];
- L[illegible] - [illegible];
- B[illegible] - [illegible];
- N[illegible] - [illegible] da 0-1;
- Tt - tempo total de CPU gasto na resolução do problema;
- Tpl - tempo gasto na resolução do (PL);
- Tpc - tempo gasto até encontrar a melhor solução da heurística pivot e complemento;
- Ttb - tempo gasto pela busca tabu na tentativa de melhorar a solução;

Tamanho de lista tabu igual a 7, para as três listas.

[illegible] algoritmo tornou-se um pouco mais lento na resolução dos problemas pequenos. Isto [illegible] do fato [illegible] cada [illegible] são reordenados (como o problema é pequeno, o tempo de [illegible], geralmente supera o tempo gasto no esquema exaustivo de busca das alternativas). Em compensação, em problemas grandes o algoritmo modificado teve um desempenho muito superior em relação à versão anterior.

Quanto a qualidade das soluções obtidas verificou-se que, para o mesmo número de iterações realizadas, o algoritmo sem melhorias mostrou, para os problemas pequenos, um [illegible], semelhante ao algoritmo inicial. Para problemas grandes, o ganho de qualidade foi relativamente superior, com esforços computacionais muito menores. O tempo gasto pelo algoritmo com melhoria equivale, em média, a 5,7 % do tempo gasto pelo algoritmo sem melhorias para a resolução dos mesmos problemas, utilizando o mesmo tamanho de lista (usou-se um tamanho de lista igual a 7, para as três listas).

Uma análise melhor dos resultados permite fazer as seguintes comparações entre as duas versões do algoritmo:

a) O [illegible]

torno de [illegible]

ii) As soluções encontradas pelo algoritmo com melhorias estão, na média, 16,11 % mais próximas do valor ótimo do que as soluções encontradas pelo algoritmo sem melhorias.

iii) Na execução das 7 últimas instâncias da série P, as soluções achadas pelo algoritmo com melhorias estão sempre abaixo de 1 % do valor ótimo, mesmo para os problemas da série PSM e aqueles para os quais a comparação é feita em relação ao ótimo do CPLD.

iv) O número de variáveis fixadas com a heurística tabu é 27,8 % maior, gerando soluções de melhor qualidade que a heurística de Balas e Martin.

Como se trabalhou com o mesmo sistema e a mesma máquina tanto na versão com melhorias quanto na sem, vê-se que o algoritmo com melhorias se mostrou bastante superior ao sem melhorias, principalmente nos problemas considerados grandes e/ou difíceis.

Algumas considerações devem ser feitas com relação aos tempos computacionais, que ainda são elevados, se comparados aos códigos comerciais existentes. Quanto a isto pode-se dizer que:

i) A fase tabu consome a maior parte do tempo computacional, principalmente quando o problema é grande e/ou difícil, não pequenas gastas [illegible] dispendiosos.

ii) Para problemas [illegible] (até 60 variáveis) os tempos são relativamente [illegible] o tempo [illegible] nas [illegible] a cada iteração não compensa [illegible] dos que ela provêm.

iii) Os tempos de CPU podem ser reduzidos ainda mais, se certas rotinas utilizadas fossem refeitas. Várias das rotinas foram construídas para uso didático e são muito ineficientes.

iv) Todos os tempos computacionais foram medidos em relação a 50 iterações do método, porém a melhor solução encontrada pode ser alcançada bem antes deste limite, o que sugere um teste de parada que seria a diferença entre o ZPL e o melhor valor da função objetivo já alcançado estar dentro de um valor de tolerância pré-determinado.

4.2. ANALISE DE SENSIBILIDADE

Esta seção é dedicada a uma análise de sensibilidade do algoritmo de busca tabu em relação a alguns parâmetros.

Será analisado qual o [illegible] da função objetivo e do número de iterações [illegible] nas várias fases do algoritmo em relação ao tamanho do problema (número de variáveis e restrições), à severidade das restrições e ao tamanho das

listas tabu.

Quatro problemas foram selecionados para esta análise de sensibilidade. Suas características estão descritas na Tabela 4.2.

NOME	m	n	Zot
P1	10	15	4015
P2	35	50	167,443
P3	50	100	730,087
P4	90	200	1518,14

TABELA 4.2 - DADOS RELATIVOS AOS PROBLEMAS

onde: m - número de restrições;

n - número de variáveis 0-1;

Zot - para P1 corresponde ao valor da solução ótima do problema 0-1,
para os restantes corresponde ao valor da solução ótima do problema linear relaxado, já que se desconhecem suas soluções ótimas.

Destes 4 problemas testados, P1 corresponde ao problema PETER3 (ver PETERSEN [21]), já P2, P3 e P4 são gerados [illegible] da mesma maneira já [illegible] no capítulo 2. Não [illegible] [illegible].

A primeira análise foi feita sobre a variação do número

de iterações máximo permitido (foram testes com 20, 50 e 100 iterações) e o tamanho das listas tabu (tamanho de lista igual a 3, 7 e 15).

O tamanho de lista foi variado simultaneamente para as três lista tabu, relativas aos movimentos simples, duplos e triplos.

Os resultados obtidos em tempos de CPU, estão mostrados nas tabelas 4.3, 4.4, 4.5, os resultados do desempenho do algoritmo aparecem nas tabelas 4.6, 4.7 e 4.8.

NOME	TAMANHO LT. TABU	Tt	Tmed	$\frac{Tpl}{Tt} \times 100\%$	$\frac{Tpc}{Tt} \times 100\%$	$\frac{Tlb}{Tt} \times 100\%$
P1	3	6,7	0,29	9,0	14,9	76,1
P1	7	7,4	0,30	6,8	13,5	79,7
P1	15	7,7	0,31	6,5	14,3	79,2
P2	3	77,5	2,10	43,8	16,1	39,1
P2	7	79,9	1,94	44,2	16,6	39,2
P2	15	75,4	2,00	44,7	16,9	38,4
P3	3	199,1	6,00	30,0	21,1	48,9
P3	7	200,2	7,05	29,5	21,0	49,5
P3	15	204,7	7,28	29,9	20,7	50,4
P4	3	1259,4	97,94	21,3	19,3	59,5
P4	7	1049,2	95,55	25,5	23,6	50,9
P4	15	1163,4	90,42	23,0	21,3	55,7

TABELA 4.3 - [illegible]

NOME	TAMANHO LT. TABU	Tt	Tmed	$\frac{Tpl}{Tt} \times 100\%$	$\frac{Tpc}{Tt} \times 100\%$	$\frac{Ttb}{Tt} \times 100\%$
P1	3	14,6	0,20	2,4	9,2	88,4
P1	7	16,2	0,21	3,2	7,1	89,6
P1	15	15,7	0,30	3,2	6,4	90,4
P2	3	127,4	1,70	22,5	10,7	64,4
P2	7	191,2	1,91	27,1	10,2	62,7
P2	15	141,6	2,04	25,1	8,6	66,3
P3	3	347,1	5,27	12,0	11,3	76,2
P3	7	404,2	6,51	14,6	10,2	75,2
P3	15	450,1	7,28	13,1	9,2	77,7
P4	3	2575,6	46,25	10,4	8,1	81,5
P4	7	2251,6	38,91	11,0	10,0	77,2
P4	15	2505,3	44,53	10,3	9,5	80,2

TABELA 4.4-RESULTADOS RELATIVOS A TEMPOS DE CPU PARA 50 ITERAÇÕES

NOME	TAMANHO LT. TABU	Tt	Tmed	$\frac{Tpl}{Tt} \times 100\%$	$\frac{Tpc}{Tt} \times 100\%$	$\frac{Ttb}{Tt} \times 100\%$
P1	3	28,4	0,29	1,8	3,9	94,3
P1	7	29,8	0,29	1,7	3,7	94,6
P1	15	34,2	0,33	1,5	3,2	95,3
P2	3	209,8	1,71	12,0	6,2	76,8
P2	7	220,6	1,91	10,1	5,9	79,0
P2	15	269,2	2,23	13,5	5,1	91,4
P3	3	634,9	5,70	9,3	6,9	84,1
P3	7	701,6	6,43	9,4	5,9	85,2
P3	15	976,6	9,02	6,7	4,7	88,6
P4	3	5171,2	49,99	7,2	4,7	88,1
P4	7	4122,3	38,65	6,5	4,9	88,6
P4	15	4703,4	42,19	7,9	4,5	87,6

TABELA 4.5-RESULTADOS RELATIVOS A TEMPOS DE CPU PARA 100 ITERAÇÕES

onde, para as tabelas 4.3, 4.4 e 4.5, tem-se:

NOME - nome do problema;

TAMANHO

LT.TABU - tamanho das listas tabu;

Tt - tempo total de CPU gasto na resolução do problema;

Tmed - tempo médio por iteração = $\frac{Tpc + Ttb}{\text{Número de iterações feitas}}$;

Tpl - tempo gasto na resolução do CPLD;

Tpc - tempo gasto até encontrar a melhor solução da heurística pivot e complemento (Balas e Martin);

Ttb - tempo gasto pela busca tabu na tentativa de melhorar a solução;

(os tempos em segundos)

NOME	TAMANHO LT. TABU	$\frac{Zot-Zpc}{Zot} \times 100\%$	$\frac{Zot-Ztb}{Zot} \times 100\%$	FIX	RxV	N SOL
P1	3	18,431	0,249	3	5	19
P1	7	18,431	0,000	3	4	23
P1	15	18,431	0,000	3	4	23
P2	3	37,890	27,737	1	6	19
P2	7	37,890	27,737	1	5	19
P2	15	37,890	22,959	2	5	22
P3	3	12,277	8,677	0	4	19
P3	7	12,277	8,677	0	4	18
P3	15	12,277	8,677	0	4	18
P4	3	4,834	4,041	2	4	26
P4	7	4,834	4,041	2	2	22
P4	15	4,834	4,041	2	2	22

TABELA 4.6 - RESULTADOS RELATIVOS AO DESEMPENHO DO ALGORITMO PARA 20 ITERAÇÕES

NOME	TAMANHO LT. TABU	Zot-Zpc x100% / Zot	Zot-Zlb x100% / Zot	FIX	REV	M SOL
P1	3	18,431	0,000	3	15	29
P1	7	18,431	0,000	3	12	23
P1	15	18,431	0,000	3	11	23
P2	3	37,800	27,139	1	15	25
P2	7	37,800	27,139	1	14	38
P2	15	37,800	22,959	2	13	22
P3	3	12,277	5,478	0	11	33
P3	7	12,277	5,744	0	13	33
P3	15	12,277	5,478	0	11	43
P4	3	4,934	3,005	2	14	26
P4	7	4,934	3,942	2	11	34
P4	15	4,934	3,512	2	12	44

TABELA 4.7 - RESULTADOS RELATIVOS AO DESEMPENHO DO ALGORITMO PARA 50 ITERAÇÕES

NOME	TAMANHO LT. TABU	Zot-Zpc x100% / Zot	Zot-Zlb x100% / Zot	FIX	REV	M SOL
P1	3	18,431	0,000	3	30	29
P1	7	18,431	0,000	3	26	23
P1	15	18,431	0,000	3	30	23
P2	3	37,800	27,139	1	34	25
P2	7	37,800	27,139	1	28	38
P2	15	37,800	22,959	2	31	22
P3	3	12,277	5,478	0	29	33
P3	7	12,277	5,744	0	26	33
P3	15	12,277	5,478	0	28	43
P4	3	4,934	3,005	2	34	26
P4	7	4,934	3,942	2	30	34
P4	15	4,934	3,512	2	29	43

TABELA 4.8 - RESULTADOS RELATIVOS AO DESEMPENHO DO ALGORITMO PARA 100 ITERAÇÕES

onde, para as tabelas 4.6, 4.7 e 4.8, tem-se:

NOME - nome do problema;

TAMANHO

LT.TABU - tamanho das listas tabu;

Zot - para P1 corresponde ao valor da solução ótima do problema 0-1,

- para os restantes corresponde a solução ótima do (PLD) relaxado;

Zpc - valor da solução 0-1 encontrada pela heurística pivot e complemento (Balas e Martin);

Ztb - valor da solução 0-1 encontrada com a aplicação de Busca Tabu;

FIX - número de variáveis 0-1 fixadas;

REV - número de movimentos reversos realizados;

M SOL - iteração em que foi encontrada a melhor solução 0-1.

4.2.1. ANALISE DO PROBLEMA P1

O problema P1 é o único, dentre os quatro problemas analisados, não gerado aleatóriamente.

Ele é um problema difícil, com poucas soluções 0-1 disponíveis, mesmo sendo um problema com um número reduzido de restrições e de variáveis 0-1.

Não é possível analisar a sensibilidade de P1 ao tamanho de lista tabu, pois ele chegou à solução ótima com todos os tamanhos de lista usados. Notase que com tamanho de lista igual a 3 ele necessitou de um maior número de iterações para chegar ao ótimo.

O problema P1 em nenhuma situação necessitou de mais de 30 iterações para chegar ao ótimo e o tempo médio por iteração se manteve constante em todos os casos.

Vêse que o algoritmo encontrou, com limite máximo de iterações igual a 80, o ótimo na iteração 23, isto acontece, porque os critérios de parada só são testados quando o algoritmo entra na fase tabu (isto é, nenhum movimento que cause um incremento na solução atual é encontrado). Portanto, o número máximo de iterações pode ser ultrapassado antes que os critérios de parada do algoritmo sejam testados.

A seguir será apresentada uma análise gráfica do problema P1, representada pelos gráficos 4.1, 4.2 e 4.3, para cada um dos três tamanhos de lista usados, mantendo fixo o número máximo de 100 iterações.

Nota-se que o problema P1, no gráfico 4.1 (tamanho de lista igual a 3) sofre um fenômeno de ciclagem a partir da iteração de número 41 e se repete a cada 10 iterações. Basta aumentar o tamanho de lista para 7 que a ciclagem desaparece

(veja gráfico 4.2). Aliás este foi o único caso de ciclagem observado.

Pode-se ver nos gráficos que o primeiro ponto de quebra corresponde ao ponto onde a heurística pivot e complemento chegou ao seu valor máximo; a partir daí a busca tabu começa a atuar na tentativa de melhorar esta solução.

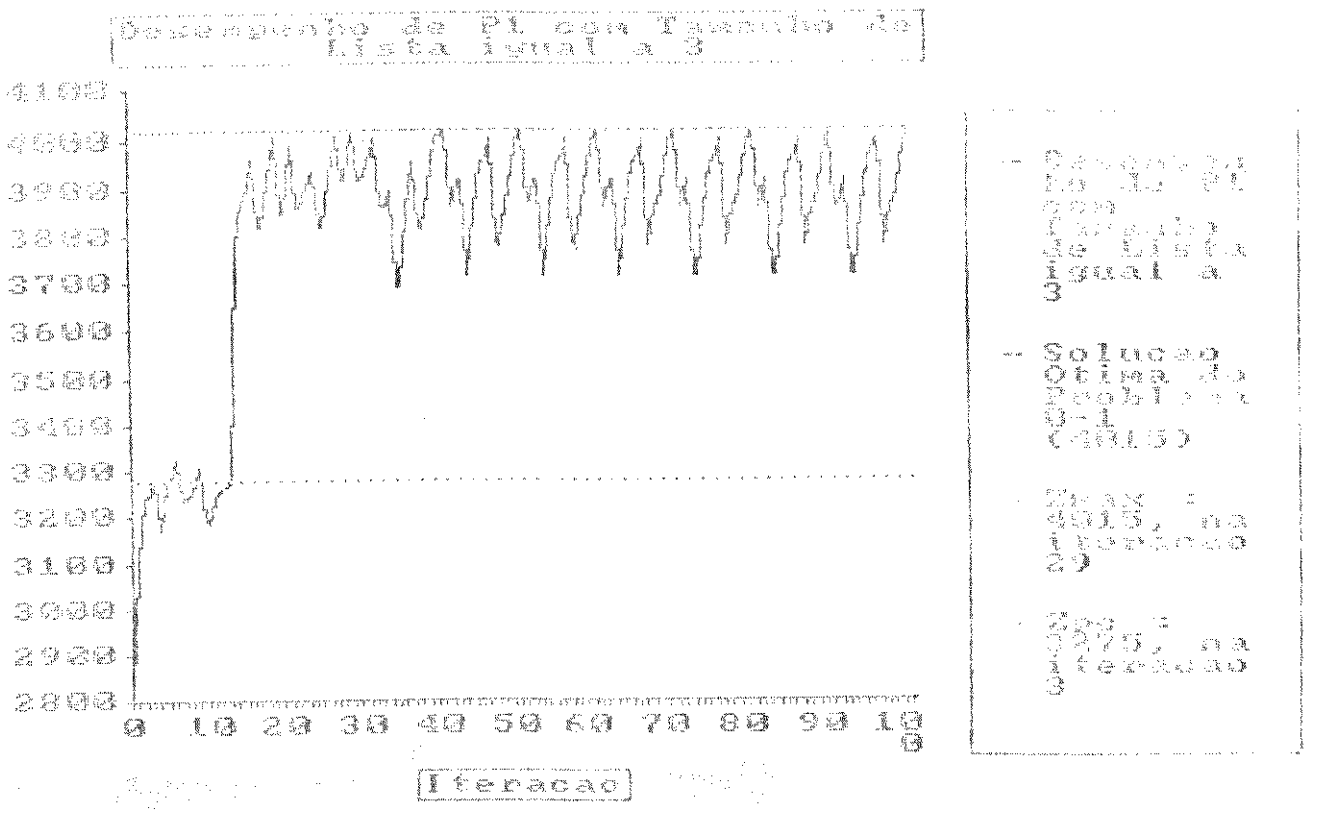
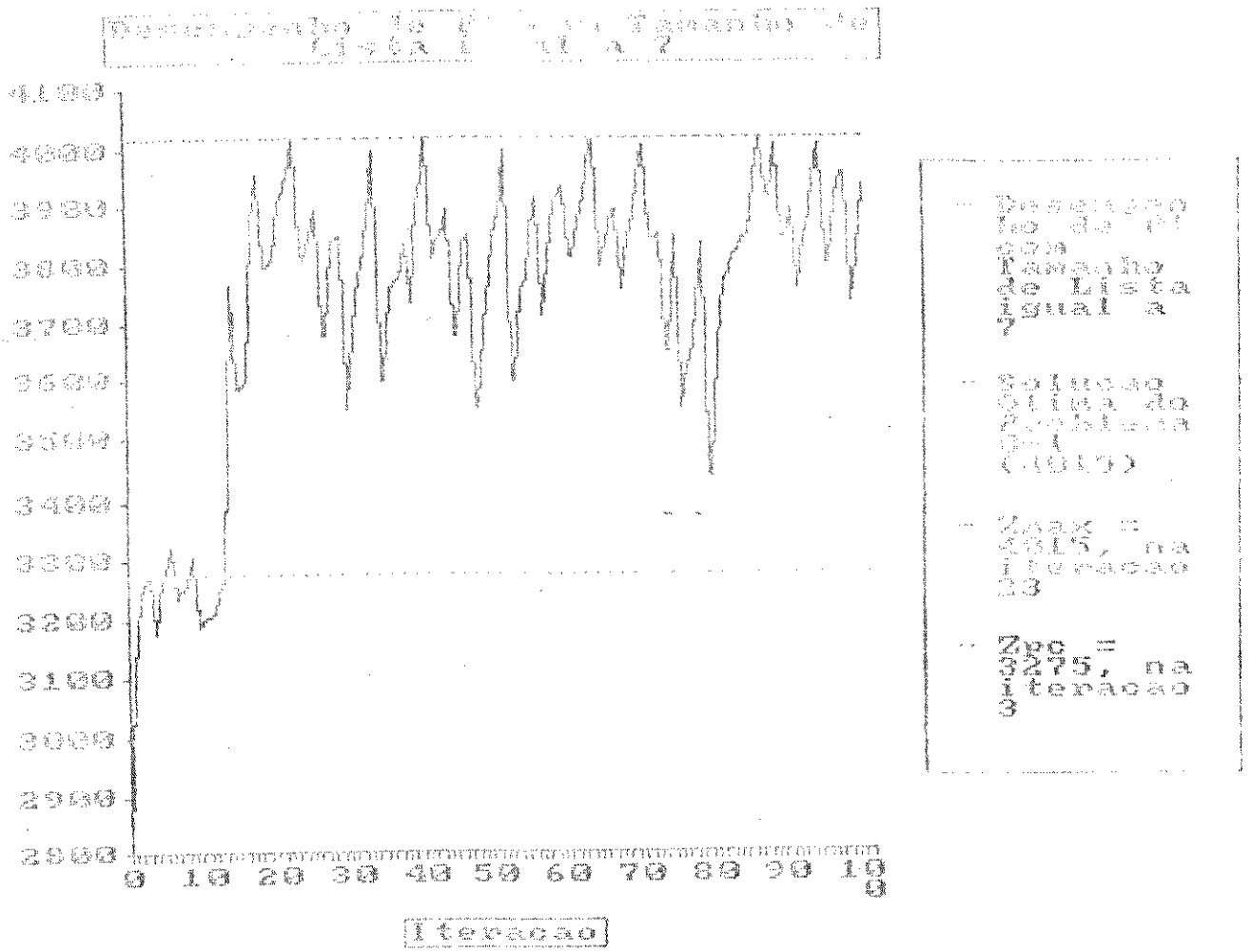
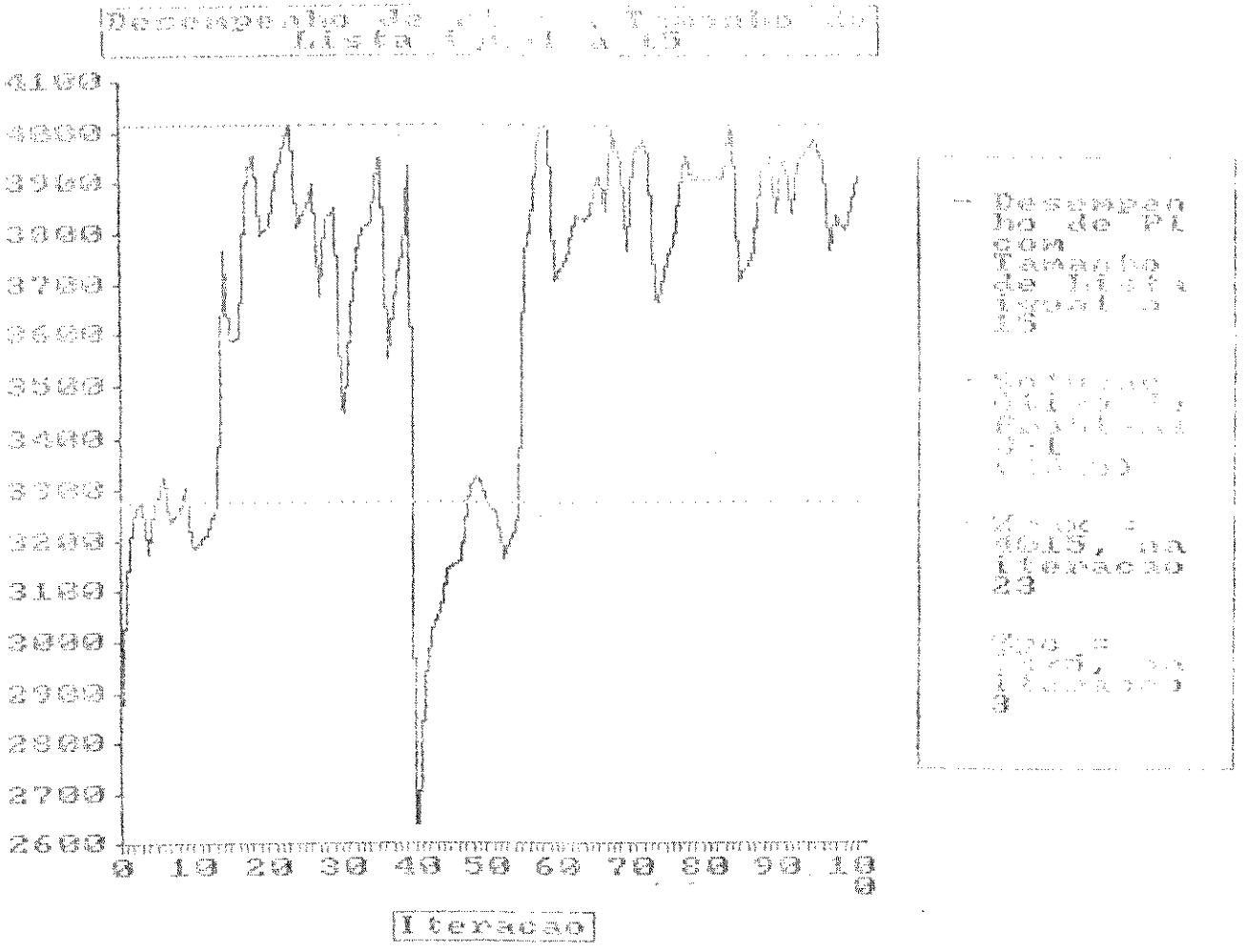


GRÁFICO 4.1


Iteracao
Tamanho da Lista = 7
Solucao Otima do Problema (4011)
Zopt = 3215, na iteracao 23
Zpc = 3275, na iteracao 3


Desempenho
Iteracao

4.2.2. ANALISE DO PROBLEMA P2

O problema P2 foi gerado aleatóriamente,tendo 50 variáveis 0-1 e 35 restrições com alta severidade, o que torna este problema de difícil solução.

P2 foi relativamente sensível ao tamanho de lista tabu, chegando a um valor de solução igual a 122 para listas com tamanhos de 3 e 7 e para a lista com tamanho 15 melhorou a solução para uma valor de 129.

O tamanho da lista também influiu na rapidez com que o processo chegou a melhor solução 0-1, pois vê-se que com tamanho de lista igual a 3 ele necessitou de 25 iterações para chegar a um valor de solução igual a 122; com tamanho de lista igual a 7 ele necessitou de 38 iterações para chegar ao mesmo valor de solução da lista de tamanho 3; enquanto isto, com tamanho de lista igual a 15 ele necessitou de apenas 22 iterações para chegar a um valor de solução igual a 129.

Nos gráficos 4.4, 4.5 e 4.6, apresentados a seguir, podemos [illegible] 100 o número de iterações máximo.

Pode-se notar que a estrutura de busca tabu começa a atuar a partir do primeiro ponto de quebra dos gráficos.

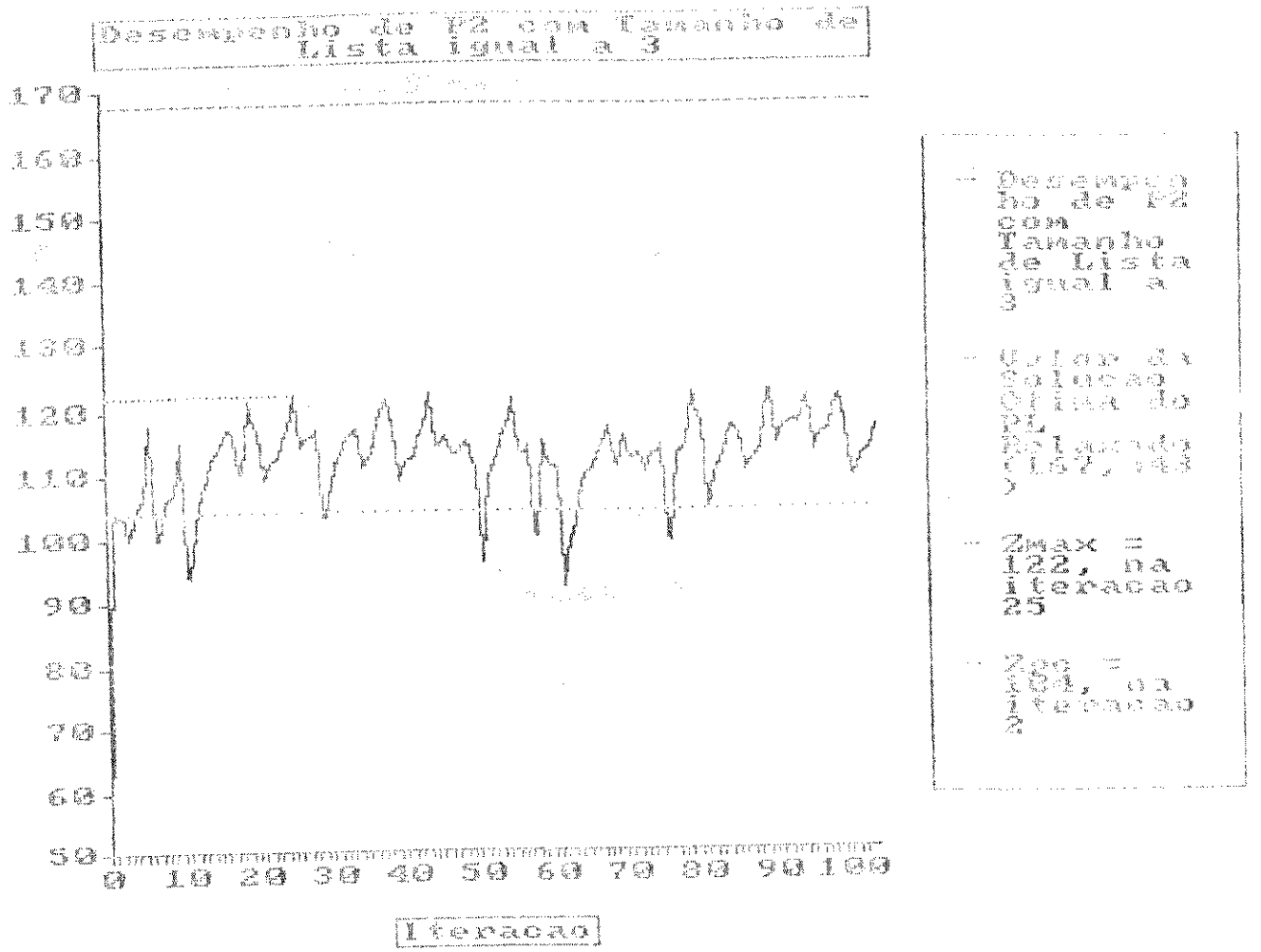
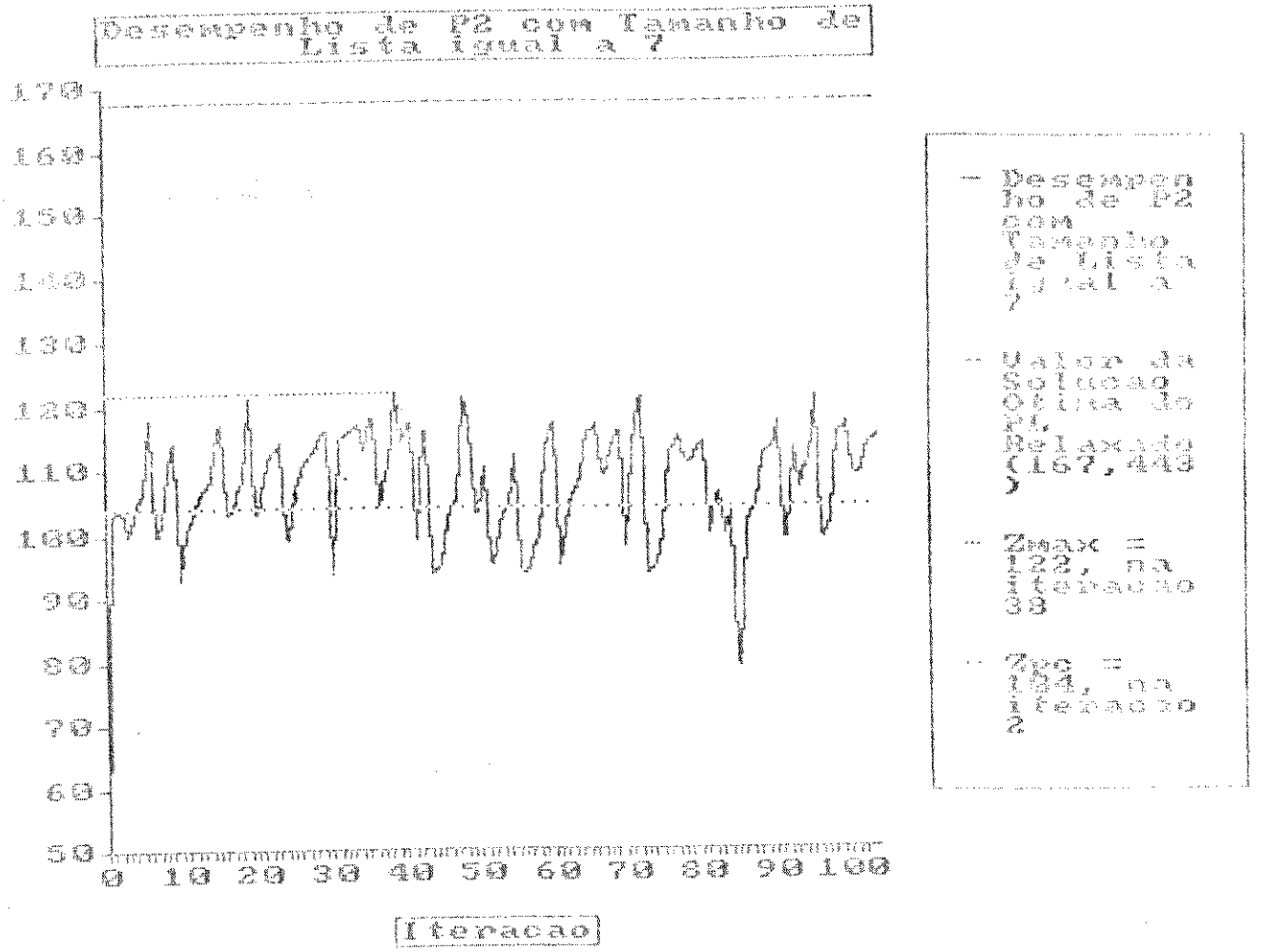


GRAFICO 4.4


Desempenho de P2 com Tamanho de Lista igual a 7
170
160
150
140
130
120
110
100
90
80
70
60
50
0 10 20 30 40 50 60 70 80 90 100
Iteracao
Desempenho de P2 com Tamanho de Lista igual a 7
Valor da Solucao Otima Relaxada (167,43)
Zmax = 122, na Iteracao 38
Zp = 154, na Iteracao 2

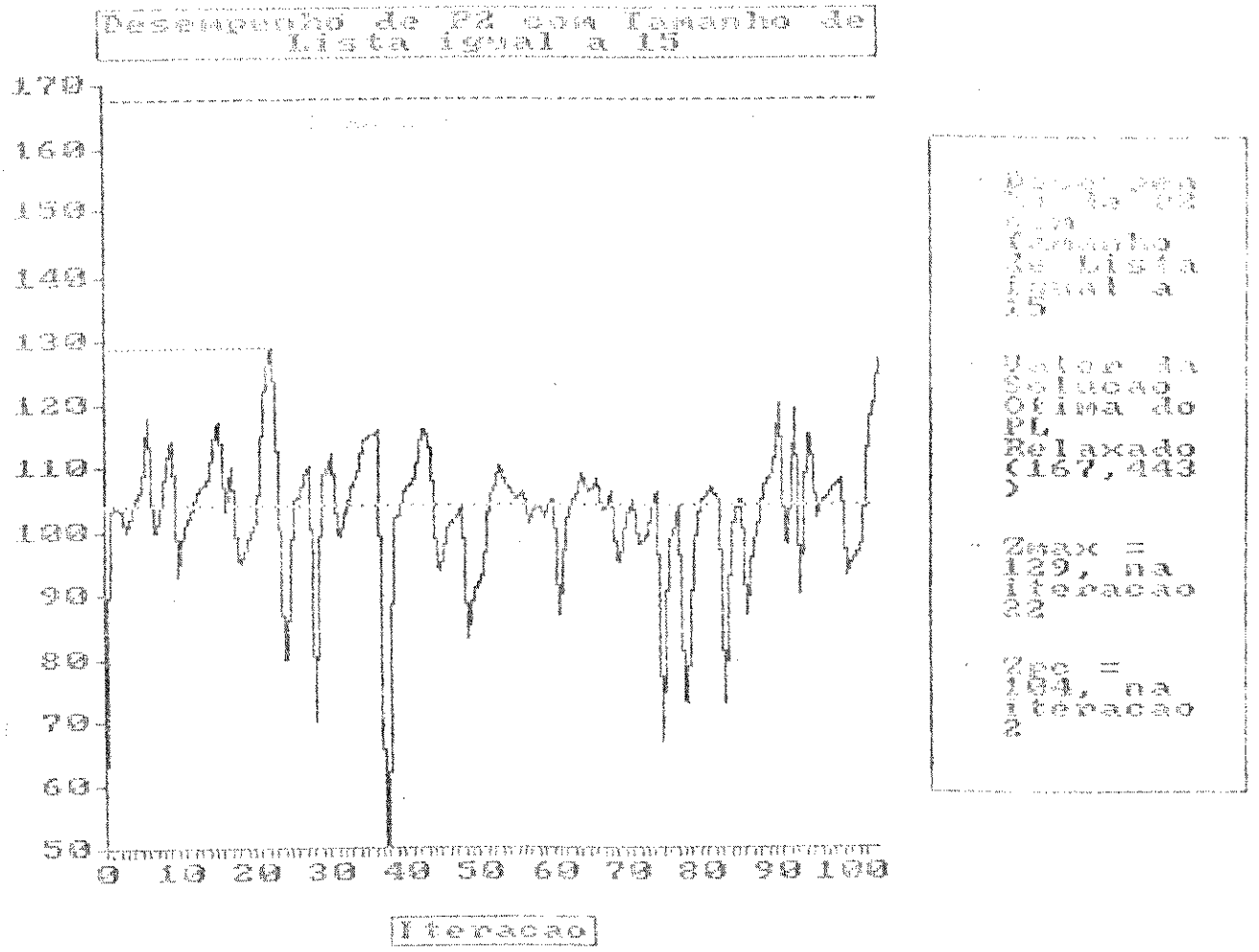

Desempenho de F2 com Tamanho de Lista igual a 15
170
160
150
140
130
120
110
100
90
80
70
60
50
0 10 20 30 40 50 60 70 80 90 100
Iteracao
Tamanho da Lista igual a 15
Valor da Solucao Otima do PL Relaxado (167,43)
Zmax = 129, na iteracao 22
Zsc = 154, na iteracao 2

Gráfico 1.6

4.2.3. ANALISE DO PROBLEMA P3

O problema P3 foi gerado aleatóriamente, tendo 100 variáveis 0-1 e 30 restrições de alta severidade, o que o torna um problema além de grande, difícil.

P3 se mostrou pouco sensível em relação ao tamanho das listas tabu e a sua melhor solução ficou oscilando entre 707 (tamanho de lista igual a 7) e 709 (tamanho de lista igual a 3 e 15).

Nota-se que o algoritmo, usando tamanho de lista gual a 15, necessitou de 15 iterações a mais do que usando tamanho de lista igual a 3, para chegar ao mesmo valor de solução (709). O algoritmo com tamanho de lista igual a 15 tem menos movimentos possíveis à sua disposição, e portanto, teve que realizar mais iterações intermediárias.

Nos gráficos 4.7, 4.8 e 4.9, apresentados a seguir, pode-se notar o comportamento iteração por iteração do problema P3 para cada um dos tamanhos de lista usados, mantendo-se fixo em 100 o número de iterações máximo.

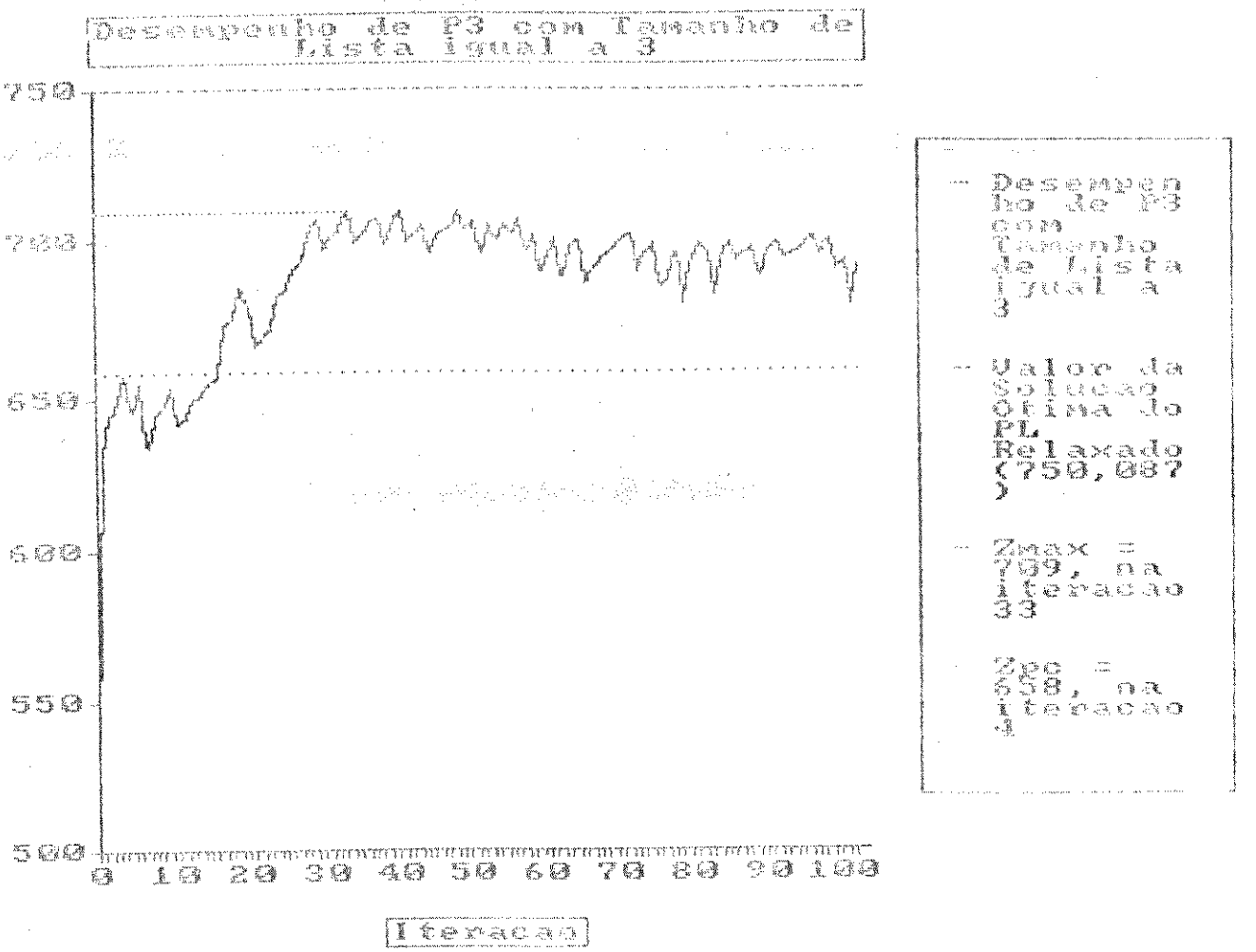
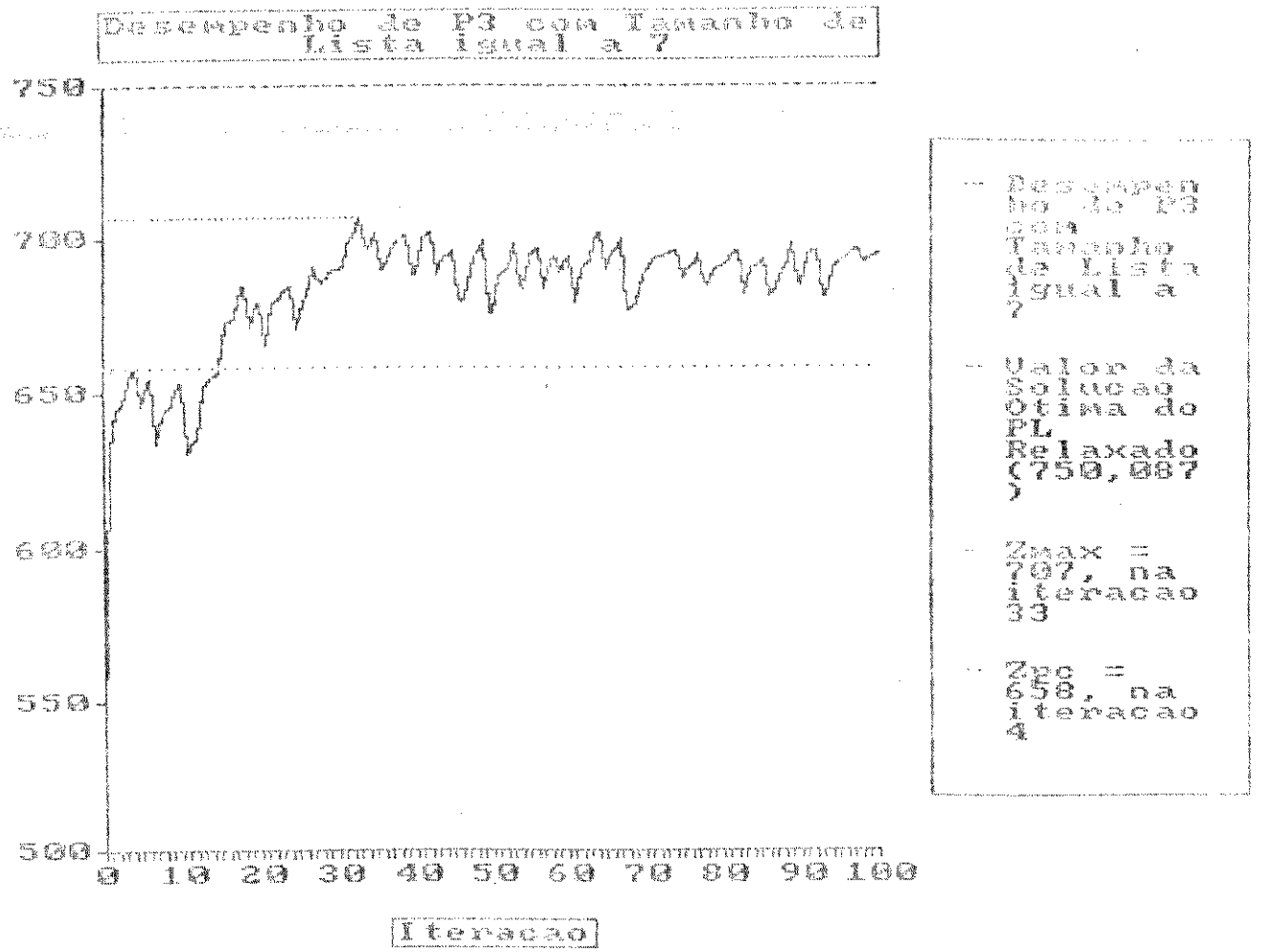

Desempenho de P3 com Tamanho de Lista igual a 3
750
700
650
600
550
500
0 10 20 30 40 50 60 70 80 90 100
Iteracao
Desempenho de P3 com Tamanho de Lista igual a 3
Valor da Solucao Otima do PL Relaxado (750,087)
Zmax = 709, na iteracao 33
Zpc = 658, na iteracao 3

GRAFICO 4.7


Desempenho de P3 com Tamanho de Lista igual a 7
750
700
650
600
550
500
0 10 20 30 40 50 60 70 80 90 100
Iteracao
Desempenho de P3 com Tamanho da Lista igual a 7
Valor da Solucao Otima do PL Relaxado (750,087)
Zmax = 707, na iteracao 33
Zpc = 658, na iteracao 4

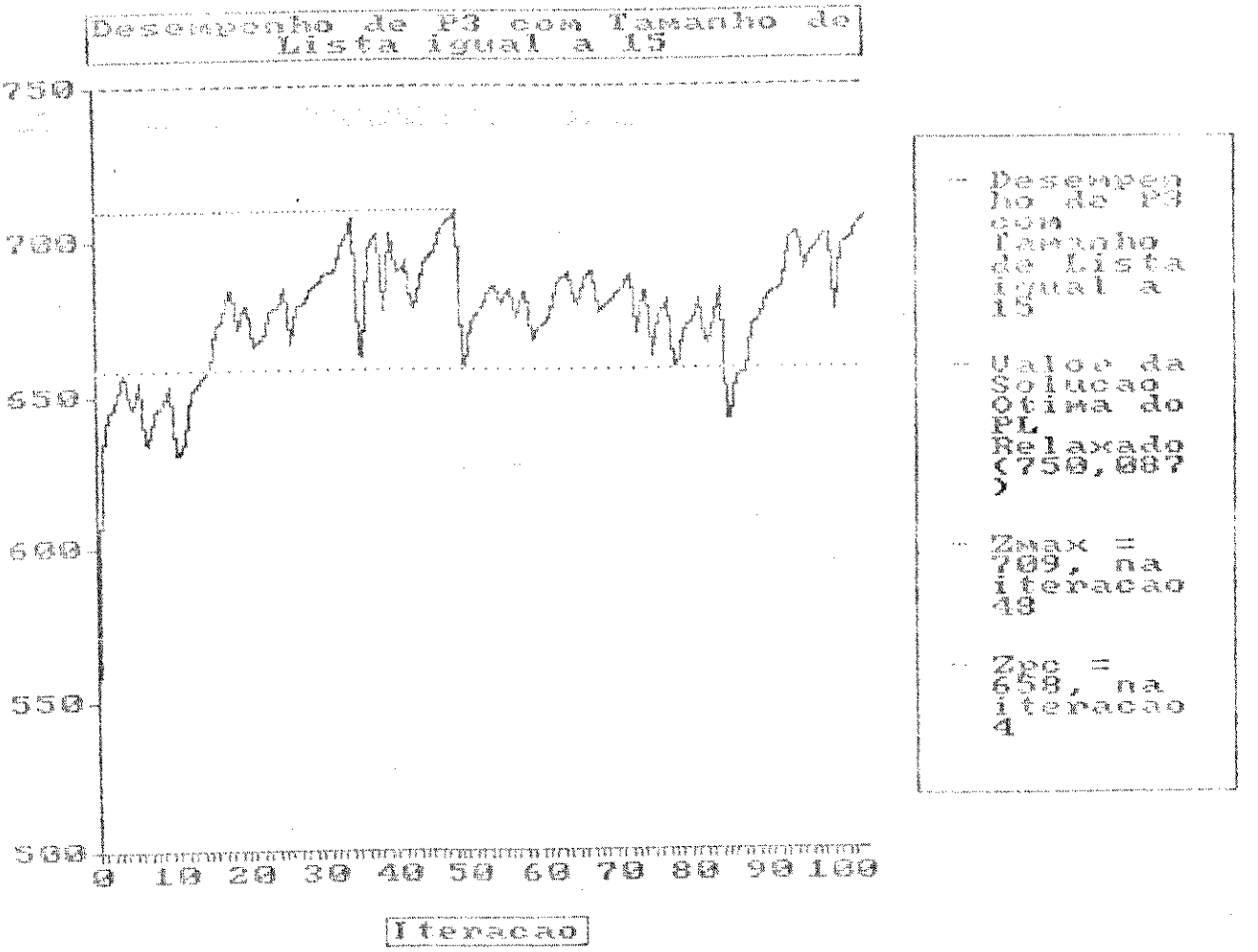

Desempenho de P3 com Tamanho de Lista igual a 15
750
700
650
600
550
500
0 10 20 30 40 50 60 70 80 90 100
Iteracao
Desempenho de P3 com Tamanho de Lista igual a 15
Valor da Solucao Ótima do PL Relaxado (750,887)
Zmax = 709, na iteracao 48
Zpc = 658, na iteracao 4

Gráfico 4.2

4.2.4. ANALISE DE P4

O problema P4 foi gerado aleatóriamente. Tendo 200 variáveis 0-1 e 30 restrições de alta severidade é considerado um problema grande e difícil.

P4 mostrou-se sensível ao tamanho de lista tabu, pois a medida que este tamanho aumenta a qualidade da solução melhora, requisitando apenas algumas iterações a mais.

Pode-se ver que o algoritmo usando tamanho de lista tabu igual a 3, encontrou a melhor solução, de valor igual a 1453, na iteração 26; usando tamanho de lista tabu igual a 7, encontrou a melhor solução, de valor 1455, na iteração 34; já usando tamanho de lista tabu igual a 15, encontrou a melhor solução, de valor igual a 1460, na iteração 43.

Nos gráficos 4.10, 4.11 e 4.12, apresentados a seguir, pode-se notar o comportamento iteração por iteração do problema P4 para cada um dos tamanhos de lista usados, mantendo-se fixo em 100 o número de iterações [illegible].

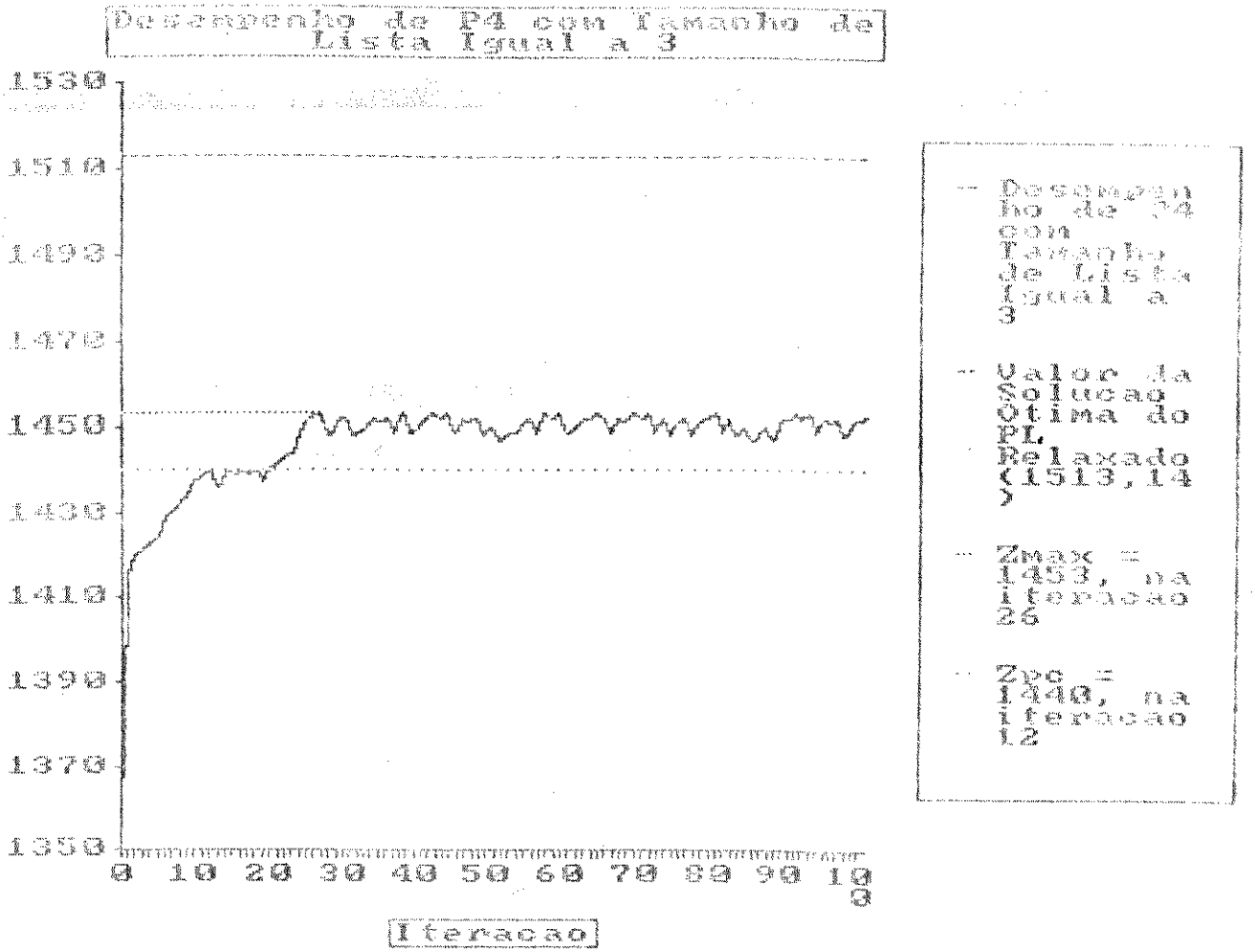

Desempenho de P4 com Tamanho de Lista Igual a 3
1530
1510
1490
1470
1450
1430
1410
1390
1370
1350
0 10 20 30 40 50 60 70 80 90 100
Iteracao
Desempenho de P4 com Tamanho de Lista Igual a 3
Valor da Solucao Otima do PL Relaxado (1513,14)
Zmax = 1453, na iteracao 26
Zpc = 1440, na iteracao 12

GRÁFICO 4.10

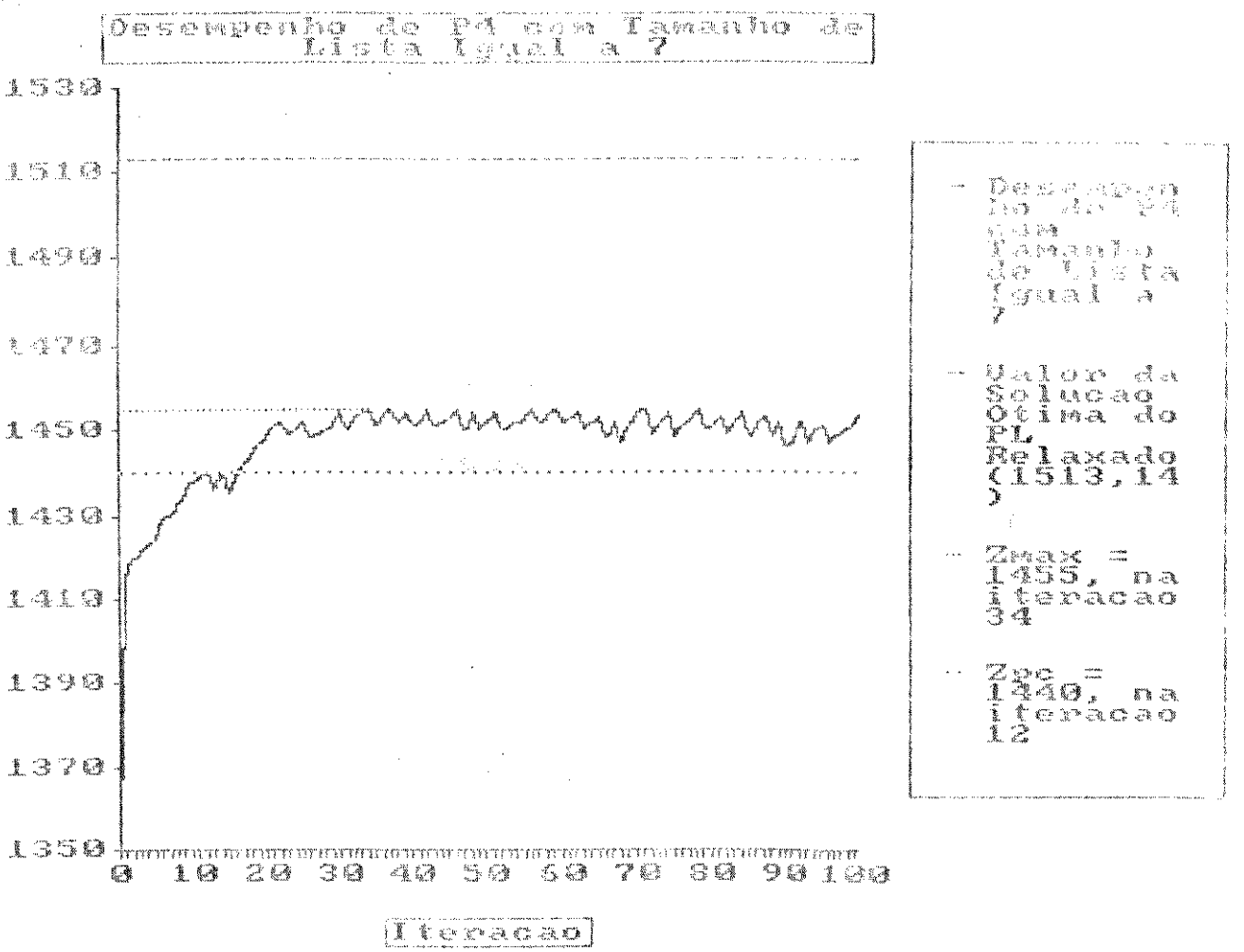
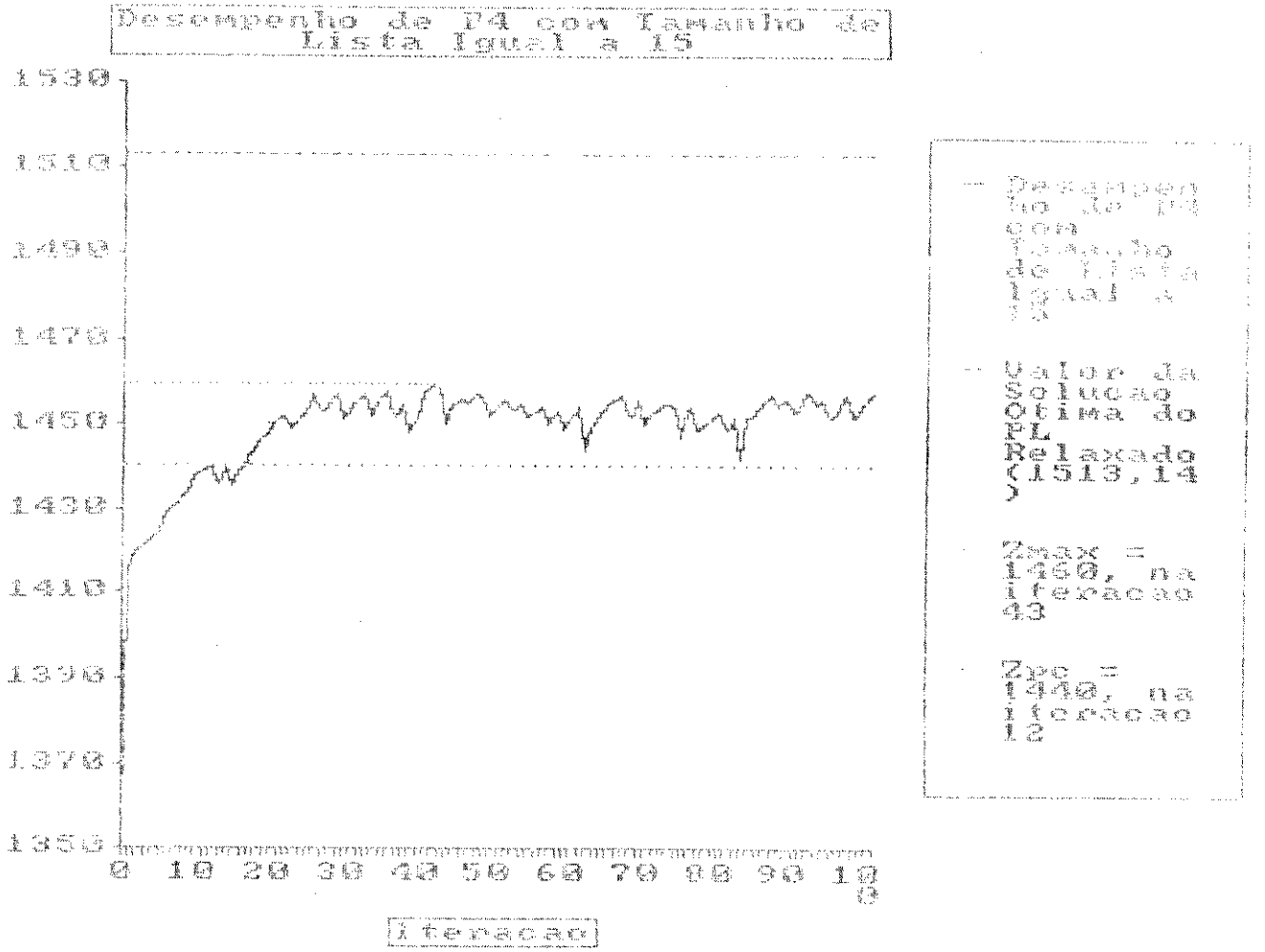

Desempenho de PL com Tamanho de Lista Igual a 7
1530
1510
1490
1470
1450
1430
1410
1390
1370
1350
0 10 20 30 40 50 60 70 80 90 100
Iteracao
- Desempenho de PL com Tamanho de Lista Igual a 7
- Valor da Solucao Otima do PL Relaxado (1513,14)
- Zmax = 1455, na iteracao 34
- Zpe = 1440, na iteracao 12

GRÁFICO 4.11


Desempenho de PA com tamanho de lista igual a 15
1530
1510
1490
1470
1450
1430
1410
1390
1370
1350
0 10 20 30 40 50 60 70 80 90 100
Iteracao
Desempenho de PA com tamanho de lista igual a 15
Valor da Solucao Otima do PL Relaxado (1513,14)
Zmax = 1450, na iteracao 43
Zpc = 1448, na iteracao 12

4.2.5. ANALISE GERAL DO COMPORTAMENTO DO MÉTODO

Em relação ao comportamento do tempo de CPU do algoritmo em função da variação do número de iterações máximo, a análise das tabelas 4.3, 4.4 e 4.5 permite afirmar que existe uma relação praticamente linear entre eles. Isto pode ser constatado através da pequena variação sofrida pela média de tempo por iteração quando aumenta-se o número de iterações.

Esta observação é válida para todos os 4 problemas e para todos os 3 tamanhos de lista, com exceções para os problemas P2 e P3 com 20 e 50 iterações máximas e para o problema P4.

Tal comportamento já era esperado, pois os esforços computacionais nas fases de melhoria e tabu são semelhantes e também porque o tempo consumido na fase de busca é pequeno em relação ao tempo total.

A confrontação do tempo de CPU com o aumento do tamanho das listas ensina que o tempo é pouco afetado quando a lista cresce. O aumento de tempo é explicado pelo maior esforço dispendido na consulta à proibição de um movimento quando se percorrem listas tabu maiores. Esta constatação permite concluir que uma decisão de usar listas maiores (por exemplo, para evitar ciclagens) não irá influenciar no tempo de execução do algoritmo.

Em relação ao comportamento do valor da função objetivo em função do tamanho das listas (tabelas 4.6, 4.7 e 4.8) é possível escrever o seguinte quadro qualitativo de tendências.

	P1	P2	P3	P4
Tamanho da Lista	↑	↑	↑	↑
$\frac{Zot - Zib}{Zot}$	→	↓	?	↓
M SOL	↓	?	↑	↑

TABELA 4.9 - QUADRO COMPARATIVO

Verifica-se que quando a lista aumenta a tendência do algoritmo é achar soluções melhores (exceção para P3). Para P1 consegue-se até, que a solução ótima seja achada em menos iterações. Já para os problemas maiores verifica-se que as melhores soluções são atingidas com um maior número de iterações. Entretanto, é preciso salientar que o aumento relativo no número de iterações para a melhor solução é pequeno se comparado com a diminuição relativa conseguida em valor de função, relembrando que ganhos relativos em regiões próximas do ótimo são custosos, especialmente para problemas muito restritos.

Assim, não parece impeditivo se se tiver que usar listas maiores; pelo contrário, há ganhos em valor de função.

Outras observações gerais podem ser feitas:

Pode-se ver que nenhum dos problemas testados necessitou de mais de 50 iterações para chegar à melhor solução 0-1, ficando este número, na média, em 32 iterações. Apenas em poucas situações (ou seja, o problema P1 com tamanho de lista tabu igual a 7 e 15 e o problema P2 com tamanho de lista igual a 15) o algoritmo chegou à melhor solução com limite máximo de iterações igual a 20.

Portanto, pode-se observar que, de uma forma geral, o melhor desempenho do algoritmo se deu quando o tamanho de lista era igual a 15 e o número de iterações situado entre 20 e 50.

Como os problemas analisados são considerados difíceis, o número de variáveis fixadas é baixo, podendo até mesmo não existir nenhuma variável fixada, como é o caso do problema P3.

O número de movimentos reversos manteve uma proporção constante entre 25 a 30 % do número de iterações realizadas, ou seja, 25 a 30 % dos movimentos realizados são movimentos que pioram a solução em questão.

As soluções melhoraram, em média, 6,6 % em relação à melhor solução encontrada pela heurística pivot e complemento de Balas e Martin, o que representa uma melhora significativa, em se sabendo que esta heurística é reconhecidamente muito eficaz.

CONCLUSÃO

Atualmente têm-se a disposição vários algoritmos, tanto heurísticos como exatos, para a solução de problemas 0-1. Os exatos chegam à solução ótima, mas muitas vezes o esforço computacional dispendido para encontrá-la não compensa. Por outro lado, os algoritmos heurísticos são muito eficazes e , geralmente com um pequeno esforço computacional chegam a uma solução factível, mas pode ocorrer que a solução alcançada seja muito pobre, ficando muito aquém da ótima (ou de uma desejável).

Neste trabalho foi feita uma tentativa de melhorar o desempenho de heurísticas para programação 0-1 através da introdução de conceitos de busca tabu.

Inicialmente, foi tentada uma adaptação direta e simples das técnicas de busca tabu na heurística clássica de Pivot e Complemento. Os resultados obtidos em relação à qualidade da solução foram animadores, mas a medida que o tamanho dos problemas aumentava, o esforço computacional dispendido tornava-se muito grande, fugindo do objetivo desejado.

Passou-se, então, a trabalhar no sentido de melhorar o desempenho do algoritmo. Para isto foram incluídos testes mais eficientes, estruturas de dados especiais e uma ordenação conveniente das variáveis.

Com isto obteve-se a flexibilidade que faltava, pois a busca da solução pôde ser feita de maneira mais inteligente, evitando a realização de movimentos pobres e também evitando a exploração de áreas não muito promissoras.

Novos testes constataram que o método melhorado não apresentava ganhos em problemas considerados pequenos (até 50 variáveis) e com restrições pouco severas, pois o tempo gasto nas ordenações a cada iteração não compensava os ganhos que dela provinham.

Em contrapartida, para os problemas grandes (mais de uma centena de variáveis) e severos o método ficou bem mais poderoso, melhorando consideravelmente a qualidade das soluções e com um esforço computacional bem reduzido, atingindo o objetivo de eficiência e eficácia.

De uma maneira geral, em todos os testes realizados o algoritmo com busca tabu melhorou (ou, poucos casos ficou igual) as soluções obtidas pelo algoritmo de Balas e Martin.

Outra consideração a fazer é que a fase tabu consumiu

grande parte do tempo de CPU gasto, mas deve-se levar em conta o fato de que ela atua em regiões muito próximas do ótimo, onde pequenos ganhos relativos são, via de regra, muito dispendiosos. Em alguns casos isto pode ser visto como uma vantagem, pois permite que se analisem várias soluções intermediárias factíveis e de valor razoável.

Não foram observados problemas de ciclagem, com exceção de um dos problemas testados. Isto mostra que uma regulagem conveniente nos parâmetros da busca tabu (tamanho de lista e número máximo de iterações) deve evitar o fenômeno da ciclagem.

Esta observação nos leva a considerar bastante auspiciosa a adaptação da busca tabu para métodos de programação 0-1, pois o fenômeno da ciclagem tem se constituido num sério obstáculo ao uso de busca tabu em alguns problemas típicos de programação matemática.

É preciso lembrar que os tempos de CPU podem ser reduzidos com uma codificação mais eficiente de certas rotinas. Não se investiu o suficiente em eficiência computacional, tendo sido aproveitadas várias rotinas construídas para uso didático.

Finalizando, pode-se dizer que, mesmo com poucas experiências computacionais realizadas, a adaptação da busca tabu à heurística de Pivot e Complemento delineia um método mais poderoso, especialmente para problemas bastante restritivos. É

necessário salientar que a versão de busca tabu utilizada neste trabalho é bastante simples. Acredita-se que a utilização de outras potencialidades do método como nível de aspiração, movimentos deficientes e dominantes e memórias de médio e longo prazo, além de uma codificação sofisticada, podem melhorar substancialmente o desempenho geral do método.

APÊNDICE

FUNDAMENTOS DE BUSCA TABU

1. BUSCA TABU

Este apêndice fornecerá as principais características da técnica conhecida por busca tabu. Para maiores detalhes ver GLOVER [12] e [15].

Busca tabu é uma técnica para resolver problemas de otimização combinatória que vão desde teoria de grafos e matróides até problemas de programação inteira pura e mista. É um procedimento adptativo que, fazendo uso de muitos outros métodos (tais como, algoritmos de programação linear e heurísticas especializadas), busca superar as limitações da otimalidade local.

Busca tabu se originou em procedimentos combinatórios aplicados a problemas não lineares de *covering* no fim dos anos 70, subseqüentemente foi aplicada em diversos problemas, indo de *scheduling* e balanceamento de canais de computadores até análise

de grupos e planejamento de espaço. Pesquisas recentes e comparações computacionais envolvendo vários problemas de otimização têm mostrado a habilidade da busca tabu em obter soluções de alta qualidade com um pequeno esforço computacional.

2. NOTAÇÃO

Para descrever o funcionamento da busca tabu, considere um problema de otimização combinatória na seguinte forma:

$$(P) \quad \text{Min} \quad c(x) : x \in X \text{ em } R^n$$

A função objetivo $c(x)$ pode ser linear ou não-linear e a condição $x \in X$ é imposta para restringir os componentes de x a assumirem valores discretos.

Um grande número de procedimentos, heurísticos e ótimos, para resolver problemas que podem ser escritos na forma de (P), poderão ser caracterizados convenientemente fazendo-se referência a seqüências de *MOVIMENTOS* que levam de uma solução *CANDIDATA* (um determinado $x \in X$) para outra. Define-se um movimento s como sendo um mapeamento definido sobre um subconjunto $X(s)$ de X:

$$s : X(s) \rightarrow X$$

Associado a x temos o conjunto S(x), que consiste dos movimentos $s \in S$ que podem ser aplicados a x, isto é, $S(x) = \{ s \in S : x \in X(s) \}$ (também podemos escrever $X(s) = \{ x \in X : s \in S(X) \}$).

Exemplos de Movimentos:

Para que se possa introduzir os principais conceitos de busca tabu é interessante que se faça uma breve discussão sobre a natureza dos movimentos que são relevantes no contexto que será examinado. Em muitas aplicações de interesse, se x' e x'' são elementos distintos de X(s), então $s(x') \neq s(x'')$; isto é conveniente para a classificação de movimentos nos quais, diferentes soluções são transformadas em novas soluções candidatas, também distintas. Um exemplo simples é o conjunto de movimentos entre os vértices adjacentes de um hipercubo 0-1, representado pelo movimento $s(x) = x + e_j$, onde e_j é um elemento do vetor de dimensão n, com valor 1 na posição j (Se X representa o conjunto de todos os vértices, então uma forma apropriada para X(s), neste caso, é $X(s) = \{ x \in X : x_j = 0 \}$.).

Outro exemplo é o movimento comumente encontrado em teoria de grafos e matróides, que consiste num aumento do passo definido em relação a um caminho alternativo. No contexto de otimização não-linear, o seu modelo padrão é $s(x) = x + \mu d$, onde d é um vetor direção dado (por exemplo, um gradiente generalizado), e μ é um escalar representando o tamanho do passo. Este tipo de

movimento pode mapear distintas soluções candidatas no mesmo vetor, a menos que sua classificação dependa tanto de μ quanto de d.

Um tipo importante de movimento para programação inteira mista é o movimento composto, que incrementa ou decrementa o valor de uma das variáveis inteiras e então determina o valor das variáveis contínuas resolvendo o problema linear resultante. Neste caso, usaremos como notação $x = (x_I , x_C)$ onde x_I e x_C são, respectivamente, os vetores de variáveis inteiras e contínuas, e seja:

$$X = \{ x : A_I x_I + A_C x_C = b , x \geq 0 \text{ e } x_I \text{ inteiro} \}.$$

O movimento composto em programação inteira mista pode ser expresso como um mapeamento de x' para x'' dado por $s(x'_I, x'_C) = (x''_I, x''_C)$, onde $x''_I = s_I(x'_I)$ e x''_C é uma solução ótima para:

$$\text{Min} \quad c(x''_I, x_C) \quad : \quad (x''_I, x_C) \in X.$$

Particularizando, $x''_I = s_I(x'_I)$ representa uma componente do mapeamento da forma $x''_I = x'_I + e_j$ ou $x''_I = x'_I - e_j$; e a condição $(x''_I, x_C) \in X$ identifica x_C como um elemento do conjunto $\{ x_C : A_C x_C = b - A_I x''_I , x_C \geq 0 \}$.

3. UMA FORMA SIMPLIFICADA DE BUSCA TABU

Inicialmente será apresentada uma forma simplificada de busca tabu que revela dois de seus elementos chave: a restrição da busca pela classificação de certos movimentos como proibidos (isto é, TABU) e a liberação da busca por uma função de memória de curto prazo, que proporciona um *esquecimento estratégico*. No decorrer desta discussão será detalhado o procedimento básico, apresentando a relação dual entre restrições tabu e critério de aspiração, fornecendo novas maneiras de restringir e guiar o processo de busca.

Além disso será discutido sobre o uso de funções de memória de médio e longo prazo, que operam como elementos auxiliares da função de memória de curto prazo, no sentido de diversificar a busca num sentido global e intensificar a busca em certas regiões.

É conveniente que se inicie fazendo referência a uma classe de métodos, conhecidos como heurísticas de *Subida de Montanha*, que progridem unidirecionalmente de seu ponto de partida até um ótimo local. (Como estamos trabalhando num contexto de minimização, a direção da escalada é feita de cima para baixo.)

Heurística Subida de Montanha para (P):

Passo 1 Selecione $x \in X$ como ponto de partida e vá para o Passo2.

Passo 2 Selecione algum $s \in S(x)$ tal que:

$$c(s(x)) < c(x)$$

Se não houver s possível, x é um ótimo local e o método PARA.

Caso contrário, vá para o Passo 3.

Passo 3 Faça $x \leftarrow s(x)$ e retorne ao Passo 2.

De um modo geral as heurísticas de *subida de montanha* são conceitualmente simples, mas podem ter características úteis, (e, às vezes, sutis) e englobar uma variedade de algoritmos matemáticos.

Por exemplo, se X é um conjunto de pontos extremos factíveis de um programa linear e S(x) é o conjunto de movimentos que levam do ponto x a um ponto extremo adjacente, então o método simplex (usando perturbação implícita ou explícita para evitar degenerescência) é semelhante a um procedimento de *subida de montanha*.

Da mesma forma, se X é a região dual factível para um programa linear então S(x) é o conjunto de movimentos que

determinam uma melhora no vetor gradiente pelo mapeamento de x em relação ao ponto (1,1,...,1). Escolhendo um tamanho de passo que gere um novo ponto a, preferencialmente, 0,9 da distância de x até o limite de factibilidade dual, vê-se que o método dual afim de programação linear de Karmarkar igualmente se parece com um procedimento de *subida de montanha*.

Estratégias de busca combinatória utilizam muito as informações, obtidas no decorrer da resolução, portanto, deve-se prestar atenção as potencialidades que residem dentro de tais métodos.

O principal limitante do procedimento de *subida de montanha*, em um dado problema combinatório, é que o ótimo local obtido é o seu ponto de parada quando não houver mais nenhum movimento de melhora possível, podendo não ser um ótimo global. Busca tabu guia esta heurística no sentido de continuar a exploração de um modo em que a busca não se torne confusa devido a falta de movimentos de melhora e sem voltar ao ótimo local anteriormente visitado. Considerando esta habilidade de incorporar e orientar outro procedimento, como se fosse uma subrotina, busca tabu pode ser vista como uma *meta-estratégia* (meta-heurística) na resolução de problemas combinatórios.

Agora será descrito o funcionamento do procedimento em sua forma simplificada. Um conjunto T de S é criado, sendo seus elementos chamados de movimentos tabu. Os elementos de T são

determinados por uma função não-Markoviana que usa informação da história do processo de busca, englobando um número de t iterações passadas, onde t pode ser fixo ou variável dependendo da aplicação ou estágio da busca. Os candidatos a estarem em T são escolhidos com o auxílio de uma lista de características ou referenciando-se a um conjunto de condições tabu (por exemplo, equações lineares de desigualdade ou relações lógicas) expressas indiretamente em termos da solução candidata atual x; por exemplo, fazendo T assumir a forma T = (s ∈ S : s(x) viola as condições tabu). Usando um T apropriadamente determinado e uma função de avaliação, denominada OPTIMUM, detalhada na seqüência, o procedimento pode ser descrito como segue:

Busca Tabu Simplificada:

Passo 1 Selecione um ponto de partida $x \in X$ e faça $x^* \leftarrow x$. Inicialize o contador de iterações k em 0 e inicie com T vazio.
Vá para o Passo 2.

Passo 2 Se S(x) - T é vazio, vá para o Passo 4.
Caso contrário, faça $k \leftarrow k + 1$, escolha $s_k \in S(x) - T$ tal que $s_k(x)$ = OPTIMUM (s(x) : $s \in S(x) - T$) e vá para o Passo 3.

Passo 3 Faça $x \leftarrow s_k(x)$. Se $c(x) < c(x^*)$, onde x^* representa a melhor solução até agora encontrada, faça $x^* \leftarrow x$.
Vá para o Passo 4.

Passo 4 Se já foi ultrapassado o limite de iterações (total ou desde a última melhora de x^*), ou se $S(x) - T = \phi$ tendo vindo para este passo diretamente do Passo 2, PARE.
Caso contrário, atualize T (como será mostrado a seguir) e volte ao Passo 2.

Deve-se enfatizar 3 aspectos desta versão: (1) o uso de T estabelece uma busca restrita durante o procedimento, portanto as soluções geradas dependem, criticamente, da composição de T e da maneira pela qual ele é atualizado no Passo 4; (2) o método não faz referência a condição de otimalidade local, exceto implicitamente onde um ótimo local supera a melhor solução anteriormente encontrada; (3) o melhor movimento (ao invés de um movimento de melhora) é escolhido em cada passo, aplicando o critério embutido na função OPTIMUM.

São apresentadas agora algumas estruturas de dados convenienetes na implementação do procedimento.

Inicialmente serão consideradas algumas formas da função OPTIMUM e do conjunto tabu T. Uma escolha óbvia para a função OPTIMUM é por $s_k(x)$ tal que:

$$c(s_k(x)) = \text{Mínimo} \{ c(s(x)) : s \in S(x) - T \}.$$

Os casos em que não há inclusão de um movimento em T,

se devem ao fato de que ele não violou os requisitos impostos por um conjunto de restrições de desigualdade (tais como, limites das variáveis) e a definição do conjunto S(x) pode ser feita de maneira semelhante, a solução s(x) obtida da definição de OPTIMUM, definido com base nestas considerações, pode representar o resultado da solução de um problema auxiliar de otimização. Esta possibilidade é importante em aplicações de programação inteira, que usam um método de programação linear como uma subrotina heurística.

Pela forma anterior de OPTIMUM, cada vez que se executa o Passo 2 do algoritmo simplificado de busca tabu, o x atual move-se para um s(x) que produz a maior melhora (ou, na falta de possibilidade de melhora, a menor piora) no objetivo c(x), sujeito a restrição de que somente os movimentos não tabu são permitidos. Há casos em que o conjunto S(x) - T pode ser grande, e, sendo processado item a item. Portanto ao invés de soluções auxiliares, é interessante que a função OPTIMUM esteja baseada em uma estratégia de amostragem desta região, reduzindo S(x) - T com o propósito de identificar o mínimo c(s(x)). Tomar o primeiro s existente que satisfaça c(s(x)) < c(x), mostra um exemplo de uma estratégia que usa amostragem, porém tabu, geralmente, trabalha mais agressivamente. Este fato contrasta com métodos que progridem lentamente até um ótimo local, partindo da premissa que uma progressão lenta, adequadamente regulada, fará com que o ótimo local encontrado esteja mais próximo de um global. (Mesmo nestes casos pode-se ter vantagens em integrar busca tabu.)

O fato da busca tabu proceder de uma maneira mais agressiva, é conseqüência de duas considerações. Primeiro é que muitos problemas de otimização podem ser otimamente resolvidos fazendo o *melhor movimento disponível* em cada passo, um fenômeno que vai além dos resultados bem conhecidos dos algoritmos gulosos.

A segunda consideração provém da estrutura do algoritmo, na qual a otimalidade local não representa uma barreira, portanto oferece menos restrições para que o procedimento pare neste ponto. Ao invés de gastar mais tempo em regiões cujas soluções são menos atrativas, busca tabu procura usar uma maior parte do seu tempo no trabalho de exploração de regiões onde as soluções são boas. (Nos casos onde as estratégias de progressão são lentas, podemos usar sua estrutura em conjunto com tabu para encontrar um bom ponto inicial de partida.)

A regra, que escolhe o mínimo c(s(x)), sujeita às restrições tabu, mostrou sucesso em várias aplicações. Quando este mínimo é custoso de calcular, uma aproximação pode ser feita, fazendo-se a escolha em um espaço reduzido da região total.

Uma forma, também simples, mas igualmente efetiva do conjunto T é dada por:

$$T = \langle\, s^{-1} : s = s_h \text{ para } h > k - t \,\rangle$$

onde k é o indicador do número de iterações e s^{-1} é o reverso do movimento s, isto é, $s^{-1}(s(x)) = x$. Ou seja, T é o conjunto daqueles movimentos que poderiam reverter (ou desfazer) um dos movimentos realizados nas t mais recentes iterações do processo de busca. Assim a operação de atualização de T no Passo 3 do procedimento simplificado de busca tabu, consiste em fazer $T \leftarrow T - s_{k-t}^{-1} + s_k^{-1}$ (os sinais de + e de - indicam, respectivamente, as operações de adição e deleção de elementos em um dado conjunto). Por convenção quando $k \leq t$ eliminamos o termo s_{k-t}^{-1}.

A forma acima indicada do conjunto tabu é baseada na suposição que a probabilidade de ciclagem, isto é, seguir uma seqüência de movimentos que levam a uma solução já visitada anteriormente, é uma relação inversa da distância da solução candidata atual x até a solução anterior. Se a distância é medida em termos do número de movimentos realizados (número de iterações feitas) desde que a solução anterior foi visitada, sujeita a condição de que nenhum movimento realizado permitiu o retorno a um de seus predecessores, então T é projetado para evitar a ciclagem de acordo com as considerações feitas acima.

Dentro de uma estrutura de certas regras de escolha e condições tabu, o objetivo mais geral é evitar um retorno a um estado de solução anterior, por exemplo, para uma solução

previamente visitada onde o melhor movimento (não tabu) disponível para sair desta solução é o mesmo realizado anteriormente. Neste caso, T deve tomar uma forma mais apropriada $T = T_1 \cup T_2$, onde $T_1 = \langle s_h^{-1} : h > k - t_1 \rangle$ e $T_2 = \langle s_h : h > k - t_2 \rangle$.

Assim, evitando a escolha de um movimento que represente o reverso de algum outro feito durante uma seqüência de t iterações, o procedimento progride movendo-se para longe de todos os estados de solução das t iterações anteriores (no sentido determinado pela natureza dos movimentos em S) e para um t suficientemente grande, a probabilidade de retornar a um estado solução já visitado, efetivamente desaparece. Entretanto, isto não quer dizer que o objetivo da busca tabu seja escolher um t grande. Numa perspectiva competitiva, quanto menor o valor de t, maior a liberdade de escolha que o método tem para guiar-se através das soluções que a função OPTIMUM achar preferíveis.

Na prática, T raramente toma a forma anteriormente mostrada, por duas razões. Primeiro, em alguns exemplos, quando o movimento s escolhido de S(x), a condição tabu que previne s^{-1} de ser escolhido também deve prevenir um conjunto mais amplo de movimentos, dominados por s^{-1}, de serem escolhidos (Mais adiante tratar-se-á de movimentos deficientes e dominados). Segundo, por considerações de conservação de espaço de memória e facilidade de processamento, geralmente, é desejável armazenar menos atributos do que o total requerido para caracterizar um movimento ou

solução, portanto armazena-se um número menor de atributos (que provavelmente poderão ser compartilhados com outros movimentos ou soluções). Sobre estas circunstâncias, uma lista tabu não consiste apenas dos movimentos s_h^{-1} para $h > k - t$, mas de coleções de movimentos C_h, onde cada C_h tem seus membros definidos por certos atributos, inclusos em suas condições tabu, fazendo com que as coleções de movimentos C_h contenham s_h^{-1} e outros movimentos que também satisfazem estas condições. Deste modo, T toma a seguinte forma, mais geral:

$$T = \cup C_h \; : \; h > k - t \quad (\text{onde } s_h^{-1} \in C_h)$$

Os conjuntos C_h podem conter, também, elementos diferentes de s_h^{-1} com o intuito de prevenir a ciclagem.

Outro aspecto importante em relação ao tratamento de T é o caso onde S(x) - T é vazio (no Passo 2 do procedimento simplificado de busca tabu). A atualização de T no Passo 4, (deixando de lado a atualização implícita de T, como mostrado anteriormente, que elimina a restrição tabu armazenada a t iterações antes), é realizada criando-se uma hierarquização no sentido de eliminar o menor número possível de elementos de T, na seqüência dos mais antigos para os mais novos, possibilitando que algum ou alguns movimentos readiquiram a sua condição não tabu. Podemos mostrar que tal organização hierárquica leva a um ótimo em um número finito de passos, em exemplos simples, se t puder crescer juntamente com a iteração k, entretanto, isto pode levar

a realização de movimentos supérfluos que poderiam ser evitados por refinamentos.

Exemplo:

Considere o problema de criação de uma partição ótima do conjunto de elementos E em conjuntos E_i , $i \in N = \{1,2,\ldots,n\}$. Cada elemento e de E tem um peso w(e), e cada conjunto E_i tem um peso W_i como objetivo. Definimos $w(E_i)$ como sendo o peso dos elementos de E que estão em E_i , isto é, $w(E_i) = \sum (w(e) : e \in E_i)$, onde $w(E_i) = 0$ se E_i é vazio. O objetivo é minimizar a soma dos desvios absolutos dos pesos $w(E_i)$ em relação aos pesos W_i (objetivo a ser alcançado).

$$\text{Minimize} \quad \sum (\, | \, w(E_i) - W_i \, | \, : \, i \in N \,)$$

Um conjunto razoável de movimentos que permitem a inclusão de busca tabu neste problema, partem de alguma distribuição arbitrária dos elementos nos conjuntos e usando trocas de um elemento e_i do conjunto E_i por um elemento e_j de E_j. Será permitido o caso especial onde e_i (ou e_j) é um elemento nulo (de peso 0) que propicia uma troca unilateral, simplesmente levando um elemento de um conjunto para outro.

A função OPTIMUM pode funcionar indicando a troca que produza a mudança mais favorável na minimização do objetivo (acréscimo ou decréscimo), restringindo-se aos movimentos

qualificados como não tabu. Usando este mesmo nível de simplicidade, o conjunto T nos fornece as regras de prevenção contra o reverso dos movimentos feitos nas t iterações mais recentes.

Para que se possa prevenir os movimentos reversos e, implicitamente, definir T, aplica-se duas estruturas matriciais, TABULIST E TABUSTATE. (Adaptações destas estruturas são amplamente utilizadas na maioria das aplicações de busca tabu.) TABULIST(p), p = 1,...,t, armazena os atributos escolhidos para caracterizar, de uma maneira compacta, as restrições tabu associadas aos movimentos. Estes atributos podem ser expressos em vários níveis de detalhamento. Em um extremo, toda a composição dos conjuntos $E_1,\ldots,E_n$ poderia ser armazenada antes e depois de um movimento, e isto seria usado para checar se alguma composição posteriormente encontrada, pela aplicação de algum movimento candidato, corresponderia àquela anteriormente armazenada, caracterizando este movimento candidato como tabu. Um conjunto mais simples de atributos, que se adapta bem a este problema, consiste de dois pares ordenados (i , $w(e_i)$) e (j , $w(e_j)$). Isto basta para identificar o fato que a troca associada a estes pares ordenados consiste na transferência de um elemento de peso $w(e_i)$ do conjunto E_i para o conjunto E_j e um elemento de peso $w(e_j)$ do conjunto E_j para E_i. Observamos que os elementos e_i e e_j não são diretamente identificados como parte destes atributos.

Tal nível de redução pode ter um valor estratégico. Neste caso, prevenir uma troca reversa de pesos é mais apropriado do que, simplesmente, prevenir a troca reversa de elementos. Portando uma boa implementação de busca tabu deverá embutir a possibilidade de proibir um movimento com $w(e_i) = w(e_j)$.

Os atributos $(i , w(e_i))$ e $(j , w(e_j))$ são usados para representar a restrição tabu que proibe o movimento no qual tentamos incluir um elemento de peso $w(e_i)$ no conjunto E_i ou um elemento de peso $w(e_j)$ no conjunto E_j. Podemos ver que o mesmo conjunto de atributos poderia conter outras restrições tabu. Por exemplo, $(i , w(e_i))$ e $(j , w(e_j))$, poderiam ser usados para proibir movimentos que retirassem um elemento de peso $w(e_j)$ do conjunto E_i ou um elemento de peso $w(e_i)$ do conjunto E_j. Estes mesmos atributos poderiam também ser usados para prevenir movimentos que ao mesmo tempo incluem e eliminam os elementos indicados, criando um movimento reverso. Este último tipo de condição tabu é menos restritiva, (isto é, proibe uma menor coleção de movimentos) que aquela inicialmente especificada.

TABUSTATE tem a função de fazer as restrições tabu fáceis de implementar. Neste exemplo, suponha que os elementos a serem particionados tenham r pesos diferentes, w_q , $q = 1,\ldots,r$. Então TABUSTATE toma a forma de uma matriz TABUSTATE(i,q) de dimensão nxr, onde TABUSTATE(i,q) tem um valor positivo se o elemento de peso q está proibido de ser incluído no conjunto E_i e assume o valor zero, caso contrário. Por motivos que logo serão

vistos, o valor positivo de TABUSTATE(i,q) é na verdade a posição p mais recente em TABULIST, na qual o atributo (i,q) está armazenado (com o propósito de proibir um elemento de peso w_q de ser incluido no conjunto E_i).

A implementação completa de busca tabu para este exemplo é a seguinte. TABULIST(p), que armazena 2 pares ordenados para cada valor de p = 1,...,t, é tratada como uma lista circular. Para fazer isto, atualiza-se o valor de p por p ← p + 1 toda a vez que o contador de iterações é atualizado por k ← k + 1, exceto quando o valor de p aumentaria de t para t + 1, então reinicializa-se p com valor 1. Cada armazenagem feita na posição p de TABULIST elimina todos os atributos que porventura existissem anteriormente nesta posição. Quando esta eliminação ocorre, TABUSTATE (que inicia com todos seus elementos iguais a zero) é modificado fazendo com que onde TABUSTATE(i,q) = p, seja reinicializado com valor zero.

Os pares ordenados, que identificam os elementos de TABUSTATE marcados para serem reinicializados com valor 0 são precisamente os dois pares ordenados armazenados como atributos de TABULIST(p). Existe uma exceção: é possível que depois de fazer TABUSTATE(i,q) = p (quando (i,q) é armazenado em TABULIST(p)), uma iteração posterior poderá envolver a retirada de um elemento de peso w_q do conjunto E_i, fazendo, deste modo, TABUSTATE(i,q) assumir um novo valor. Assim, basta observar se o par (i,q) armazenado em TABULIST(p) produz TABUSTATE(i,q) = p, se

isto não ocorrer, a operação de fazer TABUSTATE(i,q) = 0 não é realizada. Poder-se-á colocar um passo paralelo, que facilitará o tratamento dos níveis de aspiração, que serão descritos na seqüência.

Através das regras de atualização, em cada ponto onde a troca de elementos e_i e e_j foi avaliada como candidata a melhor movimento na iteração atual, os elementos de TABUSTATE(i,w(e_i)) e TABUSTATE(j,w(e_j)) revelarão se o movimento é tabu (levando em conta o fato de que no mínimo um dos elementos seja positivo). A checagem de TABUSTATE para ver se o movimento é tabu é feita, somente, após termos achado o primeiro ótimo local, ainda que a eliminação de um estado tabu seja realizada automaticamente com o uso do critério de aspiração.

Alguns pontos que ficaram para ser resolvidos são: a atualização que ocorre quando todos os movimentos da iteração atual são classificados como tabu (isto é, quando chegamos ao Passo 4 do procedimento simplificado de busca tabu diretamente do Passo 2 depois de descobrir que S(x) - T é vazio). A hierarquização dos movimentos tabu, anteriormente mencionada, poderá ser feita em relação aos valores dos elementos de TABUSTATE, de modo que o movimento escolhido seja aquele que produza a melhor mudança, no sentido de minimização do objetivo atendendo a restrição de estar entre os movimentos com mais alta prioridade. Cada elemento diferente de 0, de TABUSTATE, está associado implicitamente com um valor de iteração k, portanto os

elementos associados a menores valores de k devem ter prioridades mais altas, isto pode nos levar a escolher o melhor movimento eliminando apenas as restrições tabu mais antigas.

4. USOS DOS NÍVEIS DE ASPIRAÇÃO

Um ponto a destacar na busca tabu é a incorporação de uma <u>função de nível de aspiração</u> A(s,x), cujos valores dependem de um movimento s e/ou de um vetor x. Diz-se que um nível de aspiração está satisfeito se:

$$c(s(x)) < A(s,x)$$

A função de A(s,x) é proporcionar uma flexibilidade adicional na escolha de bons movimentos, permitindo que o estado tabu seja eliminado caso o nível de aspiração seja satisfeito. O objetivo é fazer esta operação de uma maneira que não se perca a habilidade de evitar ciclagem.

Para mostrar como realizar este objetivo, referenciar-se-á o movimento num sentido mais limitado, como um caso particular de mapeamento, por exemplo, falando de um movimento *de x para s(x)*. Este movimento, que depende tanto de x quanto de s para sua identificação, será chamado de <u>movimento solução-específico</u>.

Existem três níveis estratégicos para evitar ciclagem, os quais envolvem a prevenção de um movimento solução-específico de x para s(x) se:

(1) s(x) já foi anteriormente visitado;

(2) o movimento s já foi anteriormente aplicado a x;

(3) o movimento s^{-1} já foi anteriormente aplicado a s(x).

Apesar de (1) ser o único critério de prevenção de um movimento solução-específico que assegura a não ocorrência de ciclagem, o processo de checagem se o estado tabu de um movimento pode ser eliminado baseando-se em (1), geralmente, requer mais memória e mais esforço do que seja conveniente aplicar. Se um movimento tabu é permitido com base, somente na falha da condição (2), então é possível reverter um movimento tão logo ele seja realizado, retornando a uma solução já visitada. Portanto, vê-se que listas tabu construídas deste modo não trabalham bem. Entretanto, a condição (3) é compatível com uma estrutura de lista tabu. Usando a checagem da condição (3) para evitar ciclagem e permitindo a realizaçâo de um movimento tabu que leve de x para s(x), a não ser que o movimento solução-específico de s(x) para x já tenha ocorrido, no objetivo de prevenir o retorno a uma solução anteriormente gerada, o teste usando a condição (3) se mostrou mais efetivo do que o teste usando a condição (2). A

união da condição (2) à condição (3) serve para se chegar mais próximo da efetividade da condição (1).

A importância destas observações na criação de uma função efetiva de nível de aspiração, poderá ser melhor vista através de um exemplo concreto. Definindo A(s,x) como o melhor (menor) valor de c(x') que poderia ser encontrado revertendo-se um movimento solução-específico anterior a x' para algum s'(x') tal que c(s'(x')) = c(s(x)).

Embora a definição pareça um tanto complicada, o conceito mencionado é bastante simples e a função de nível de aspiração em questão é fácil de ser armazenada e atualizada. Suponha que os valores possíveis de c(x), para diferentes vetores x, são dados pelos inteiros q = 1,2,...,U e seja BEST(q) = mínimo (melhor) valor de c(x) que poderia ser alcançado pela reversão de um movimento anterior que produziu c(s(x)) = q. Inicializa-se BEST(q) = U + 1. Então, no momento em que é realizado um movimento solução-específico, atualize BEST(q), para q = c(s(x)), através da regra BEST(q) = Min (BEST(q) , c(x)). Após isto, o nível de aspiração poderá permitir a realização de um movimento tabu que leve de (um diferente) x para s(x) se:

c(s(x)) < BEST(c(x)).

Este tipo especial de aspiração é um exemplo de checagem da condição (3).

A forma correspondente da condição (2), neste exemplo, é dada definindo-se A(s,x) = o melhor (menor) valor de c(s'(x')) encontrado sobre todos os movimentos solução-específicos anteriores, partindo de algum x' para algum s'(x'), onde c(x') = c(x). Esta função de nível de aspiração alternativa pode ser desenvolvida, inicializando-se BEST(q) da mesma maneira anterior, mas quando um movimento solução-específico de x para s(x) é feito, BEST(q) é atualizado, para q = c(x), do seguinte modo:

$$BEST(q) = Min (BEST(q) , c(s(x)).$$

A condição para a realização de um movimento tabu que leve de (um diferente) x para s(x) é idêntica a anterior, ou seja:

$$c(s(x)) < BEST(c(x)).$$

As condições (2) e (3) são facilmente combinadas, fazendo-se as duas atualizações de BEST(q) como mostrado anteriormente, tendo em mente que o teste para que o estado tabu seja desprezado, é o mesmo em ambos os casos.

Outras funções de nível de aspiração que podem ser facilmente checadas e atualizadas são feitas mantendo registros semelhantes de outros atributos do movimento, inclusive pode-se

usar função de nível de aspiração múltiplas, que aumentam a probabilidade do movimento escapar de seu estado tabu (requer que no mínimo uma das inequações associadas seja válida), entretanto, exigem um maior esforço na checagem e armazenagem. Uma alternativa interessante é definir A(s,x) somente em relação aos movimentos atualmente considerados tabu, de uma maneira que será discutida na seqüência.

5. RESTRIÇÕES TABU INTEGRADAS COM CRITÉRIOS DE NÍVEL DE ASPIRAÇÃO

As restrições tabu e o critério de nível de aspiração da busca tabu desempenham um papel dual no que diz respeito a restringir e direcionar o processo de busca. Restrições tabu permitem que o movimento seja considerado admissível se elas não se aplicam, enquanto o critério de aspiração permite que o movimento seja considerado admissível se ele se aplica (isto é, seja satisfeito). Esta complementariedade dos conceitos de restrições tabu e critério de aspiração permite que se trate deles integrando-os numa estrutura comum.

Para se descrever esta estrutura, é conveniente que seja apresentada uma notação para os atributos dos movimentos que são usados para definir o estado tabu. Especificamente, $a_p(s,x)$, $p = 1,\ldots,g$, é o conjunto de funções que identificam certos atributos do movimento s aplicado a solução x. Para clarear estes

conceitos, será apresentado um procedimento que integra critério de aspiração e restrições tabu para o problema do caixeiro viajante, baseado no uso dos movimentos 2-OPT que eliminam dois arcos não adjacentes da rota e adicionam os dois únicos arcos que criam uma nova rota. Funções de atributos para esta aplicação são definidas de uma maneira bem simples, ou seja, $a_1(s,x)$ e $a_2(s,x)$ identificam os dois arcos adicionados e $a_3(s,x)$ e $a_4(s,x)$ identificam os dois arcos eliminados (tomando g = 4) da rota.

O grau de especificação na identificação de tais atributos dos movimentos pode variar. Por exemplo, $a_1(s,x)$ e $a_2(s,x)$ podem se referir aos arcos adicionados sem prestar atenção a nenhuma ordenação em particular ou pode diferenciar, mais precisamente os arcos, por exemplo $a_1(s,x)$ identifica o maior dos dois arcos adicionados e $a_2(s,x)$ identifica o menor (o mesmo ocorrendo para os arcos eliminados).

Para se implementar estas funções de atributos diz-se que E é o conjunto de todos os elementos que podem ser identificados como atributos dos movimentos. No problema do caixeiro viajante, E consiste no conjunto de todos os arcos do grafo. Pode-se identificar também outros atributos relevantes dos movimentos, neste exemplo, que fazem a natureza de E mais complexa, por exemplo, pode-se introduzir $a_5(s,x)$ e $a_6(s,x)$ para representar os valores de $c(x)$ e $c(s(x))$, no caso onde E poderia incluir, além dos arcos, o comprimento das rotas. Generalizando, neste caso, E poderia ser tratado como uma coleção de conjuntos

que contivessem os elementos para as diferentes classes de atributos (Pode-se notar que a identificação de $a_5(s,x)$ e $a_6(s,x)$ nos coloca numa posição mais próxima do tipo de critério de nível de aspiração discutido na seção anterior). Os níveis de aspiração, são definidos em relação aos elementos de E e tratados pela mesma estrutura de tempo de permanência anteriormente indicada para criação e manutenção de listas tabu.

Em particular, uma lista tabu diferente, T_p, $p = 1,\ldots,g$, é criada para cada atributo $a_p(s,x)$, podendo-se definir:

$$T_p = \{ a_p(s_h , x^h) \ : \ h > k - t_p \},$$

onde s_h e x^h se referem ao movimento s e ao vetor x na iteração h.

No exemplo do caixeiro viajante, isto corresponde à criação de uma lista tabu para cada arco na troca 2-OPT, e dar a cada lista T_p seu próprio tamanho t_p. (O menor, dos dois arcos eliminados de uma rota, deve ter um período de permanência menor na lista tabu do que o maior, isto é, permitindo-se que ele seja novamente adicionado a rota mais rapidamente).

Estado tabu, ou equivalentemente, *estado de aspiração*, neste momento é definido pela combinação destas listas tabu com uma função de aspiração A(e) definida em relação aos elementos e

de E. Assim, quando um elemento e = a_p(s,x) é um atributo do movimento de x para s(x) (por exemplo, onde e é um dos arcos adicionados a rota representada por x), o valor de A(e) é atualizado de acordo com um critério semelhante ao da seção anterior, ou seja, A(e) = Min (A(e) , c(x)) ou A(e) = Min (A(e) , c(x) , c(s(x))), etc ... . A(e) é inicializado com um valor muito grande (infinito) para cada valor e em E, e atribui-se a ele novamente este valor, toda vez que e não pertencer a mais nenhuma das listas tabu.

O estado tabu ou estado de aspiração, de todos os movimentos pode ser uma função do estado de seus atributos. Dito que o movimento candidato s aplicado a x produzirá um valor objetivo c(s(x)), é natural atribuirmos a este movimento um estado liberado (indicando que ele satisfaz o teste de aspiração) se c(s(x)) é menor que o valor aspirado para um determinado atributo. Especificamente, um atributo a_p(s,x) recebe um estado liberado se c(s(x)) < A(e) para e = a_p(s,x). Todos os elementos e, fora da lista tabu recebem automaticamente o estado liberado. Outros recebem o estado liberado se o movimento para o qual eles contribuem produz um valor objetivo superior ao valor aspirado armazenado quando eles se tornaram tabu. O movimento s ou, mais precisamente, a transição de x para s(x), pode receber um estado liberado se todos ou um número selecionado de seus atributos receberam um estado liberado ou, até mesmo, se um atributo de importância suficiente recebeu um estado liberado.

A partir desta discussão, vê-se que a representação do conjunto tabu T como uma coleção de movimentos (isto é, um subconjunto de S) no procedimento simplificado de busca tabu, na seção 3, não é adequado para grandes aplicações. Em geral, T pode ser visto como uma coleção de pares (s,x), ou seja, de movimentos solução-específicos, caracterizados por um conjunto selecionado de atributos. A especificação $s \in S(x) - T$ fica substituida pela especificação $s \in S(x)$ e $(s,x) \notin T$. (No caso em que pares (s,x) de T são capazes de identificar soluções, sem fazer referência aos movimentos aplicados a eles, a exclusão de (s,x) de T pode ser vista, de uma maneira mais simples, como a exclusão de s(x) de T).

Resumindo, critério de aspiração e restrições tabu podem ser usados obedecendo a uma estrutura organizacional comum e vista sobre diferentes aspectos do mesmo princípio conceitual. Num extremo, uma aspiração A(e) que é fixada menor do que qualquer valor de c(x) possível corresponde a um pré-estabelecimento do estado tabu. O motivo que valida esta forma de integração de critério de aspiração e restrições tabu é a hipótese que diferentes atributos dos movimentos podem ter diferentes influências em relação a qualidade das soluções geradas, sendo assim, sujeitos a diferentes tempos de duração de seu estado tabu (isto é, devem ser armazenados em listas tabu de diferentes tamanhos) e governados por diferentes niveis de aspiração, que podem ser levados a cabo pelo mesmo mecanismo geral.

6. A FUNÇÃO DE MOVIMENTOS DOMINANTES E DEFICIENTES

Dominância e equivalência agem em busca tabu de um modo diferente do que nos contextos usuais de otimização, requerendo referências a natureza do movimento aplicado. Como ilustração considere um problema binário da mochila, da seguinte forma:

$$\text{Max} \quad 7x_1 + 6x_2 + \ldots + 4x_7$$

$$\text{s.a.} \quad 4x_1 + 5x_2 + \ldots + 3x_7 \leq 20$$

Se uma condição tabu previne um movimento que substitui $x_1 = 0$ por $x_1 = 1$, então pelas considerações aparentes de dominância poder-se-ia prevenir também o movimento que substitui $x_2 = 0$ por $x_2 = 1$. Na direção contrária uma condição tabu que previne um movimento que substitui $x_2 = 1$ por $x_2 = 0$ deveria prevenir também um movimento de $x_1 = 1$ para $x_1 = 0$. Note que esta dominância, caracterizada em relação aos movimentos, não é do mesmo tipo da dominância que implica que uma variável terá valor zero na solução ótima. (Neste caso, tanto x_1 quanto x_2 podem ser iguais a 1.)

No caso em que os coeficientes de x_2 são idênticos aos de x_1, então os movimentos correspondentes a eles podem ser vistos como equivalentes, e são necessárias extensões semelhantes das restrições exigidas por tabu. Equivalência age diferentemente

de estrita dominância, entretanto, um movimento não é permitido se uma alternativa estritamente dominante está disponível. (Por exemplo, o movimento de $x_1 = 1$ para $x_1 = 0$ não será permitido se $x_2 = 1$ na solução candidata em questão). Movimentos equivalentes não trazem limitações correspondentes, embora possam ser classsificados arbitrariamente com o intuito de serem tratados do mesmo modo que movimentos estritamente dominantes.

Sabendo que a busca tabu previne a ocorrência de certos movimentos, pode-se aplicar o critério de avaliação de uma heurística padrão, dentro de OPTIMUM, o que pode levar a escolha do *melhor* movimento como sendo, um que não poderia ser realizado. Por exemplo, no problema da mochila, uma heurística troca variáveis na solução mantendo a factibilidade, substituindo $x_i = 1$ por $x_j = 1$ para um par especificado (i,j) e avaliando a troca com base na sua contribuição na função objetivo. A heurística opera sem problemas quando aplicada a um procedimento de subida de montanha, entretanto, novas circunstâncias aparecem quando a heurística inclui em seu corpo a função OPTIMUM. A avaliação da heurística deve ser modificada para proporcionar à busca tabu habilidade de achar condições não encontradas durante a subida de montanha. Especificamente, no exemplo anterior do problema da mochila, a troca que substitui $x_1 = 1$ por $x_2 = 1$ poderá ser vista como tendo um valor de avaliação mais alto (isto é, produzirá uma melhor mudança na função objetivo) que a troca que substitui $x_1 = 1$ por $x_7 = 1$. (Observe que ambos os movimentos são de piora). Busca tabu, entretanto, classifica o primeiro movimento como um

movimento deficiente, pois ele não permite nenhuma possibilidade de melhora da solução. Tais movimentos são eliminados de consideração pela OPTIMUM, sem levar em conta o fato que eles podem receber avaliações mais altas que outros movimentos da heurística subida de montanha padrão.

7. FUNÇÕES DE MEMÓRIA DE MÉDIO E LONGO PRAZO

Funções de memória de médio e longo prazo são usadas em busca tabu para obter, respectivamente, uma intensificação local e uma diversificação global da busca. Combinadas com as funções de memória de curto prazo, representadas pelas listas tabu, as funções de médio longo prazo propiciam uma interação entre *aprendizado* e *esquecimento*.

Memória de médio prazo age armazenando e comparando características de um número selecionado das melhores soluções candidatas geradas durante um dado período da busca. Características, que são comuns a todas ou dizem respeito a maioria destas soluções (tal como valores atribuidos a certas variáveis), são tomadas como atributos locais destas boas soluções. O método, então, busca novas soluções que apresentem estas características, restringindo ou penalizando movimentos disponíveis durante um período subsequente de intensificação da busca local.

A estrutura de busca tabu, leva de uma vez, a um tipo de estratégia de intensificação útil para resolver problemas grandes. Devido a isto, busca tabu tende, fortemente a concentrar-se na geração de *boas* soluções e também tende, geralmente, a incorporar somente um subconjunto de elementos de decisão (variáveis, arcos, etc,...) nestas soluções. Por exemplo, no contexto do problema do caixeiro viajante, para grafos moderadamente densos, o número de arcos diferentes incorporados nas rotas de alguma solução anterior dada é, geralmente, somente uma fração do número total de arcos. Assim, após um número inicial de iterações, o método pode deixar de lado todos aqueles arcos que ainda não foram incorporados em nenhuma rota e concentrar-se na resolução do problema reduzido resultante. Por esta razão as iterações se tornam muito mais rápidas de executar, então a busca pode examinar muito mais alternativas em um mesmo espaço de tempo considerado, bem como concentrar-se em possibilidades que aparentem ser atrativas.

Usos mais avançados de memória de médio prazo englobam a criação de uma rede de boas soluções, como uma matriz para a geração de outras soluções com boas propriedades. Em particular, um conjunto de boas soluções é usado para definir uma subregião do espaço de busca que contém estes pontos (e outras regiões próximas ou definidas por combinações convexas, etc,...) para fornecer um foco de intensificação de busca e para lançar-se em explorações das regiões vizinhas. Tal procedimento pode ser ampliado pelo uso de formas avançadas de análise de

discriminação, que buscam gerar um conjunto reduzido de inequações e condições lógicas de dois tipos:

(1) que contenham boas soluções;

(2) que separem as soluções boas das ruins.

A função de memória de longo prazo, cujo objetivo é diversificar a busca, aplicando princípios que são, de uma maneira geral, o reverso daqueles vistos para memória de médio prazo. Ao invés de induzir uma concentração da busca em regiões que contenham (ou que possamos supor que contenham) boas soluções anteriormente encontradas, a função de memória de longo prazo guia o processo em direção a regiões que são caracterizadas por grandes contrastes em relação as que já foram examinadas. Este procedimento difere dos métodos que buscam diversificação pela geração de uma série aleatória de pontos de partida e, portanto, não propiciam oportunidade de aprendizado com o passado. O objetivo é criar um critério de avaliação que possa ser usado pelo processo de busca heurística, projetado especificamente para produzir um novo ponto de partida, assim gerando um novo ponto de uma maneira mais inteligente do que gerá-lo por processos aleatórios. Este critério de avaliação penaliza as características que a memória de longo prazo encontra como predominantes nas execuções anteriores do processo de busca. Novamente, o problema do caixeiro viajante fornecerá o exemplo conveniente. Uma forma simples de memória de longo prazo, neste

exemplo, é um contador do número de vezes que cada arco aparece nas rotas geradas. Penalizando cada arco com base neste contador, está se favorecendo a geração de *boas* rotas de partida (de acordo com a heurística escolhida para produzir estas rotas) que tendem a evitar estes arcos comumentemente encontrados anteriormente. A diversificação encontrada por este critério pode ser aumentada pela retenção destas penalidades por um período depois de impostas, transferindo-as para a (possivelmente diferente) heurística associada com busca tabu e usar o procedimento de busca tabu para procurar soluções melhores. Posteriormente as penalidades são eliminadas e a busca tabu procede de acordo com seu critério de avaliação normal. O mesmo tipo de procedimento pode ser usado para continuar, diretamente do ponto atual da busca, para uma nova região, sem voltar atrás para gerar uma nova solução sem conhecimentos prévios. Uma maneira de implementar esta estratégia de longo prazo é através de uma lista tabu de longo prazo, periodicamente ativada, que pode aplicar um critério de melhora muito rigoroso que leve o processo de solução em direção a um território não explorado.

[6] CROWDER, H. JOHNSON, E.L. PADBERG, M.W. Solving large-scale 0-1 linear programming problems, Operations Research, Vol.31(4), pp. 803-934, 1983.

[7] FERLAND, J.A. FLORIAN, M. A sub-optimal algorithm to solve large scale 0-1 programming problem, Survey of Mathematical Programming II, pp. 461-469, 1975.

[8] GARFINKEL, R.S. NENHAUSER, G.L. Integer Programming, John Wiley and Sons, 1ª ed., 427 p., 1972.

[9] GEOFFRION, A.M. An improved implicit enumeration approach for integer programming, Operations Research, Vol.17, pp. 437-454, 1969.

[10] GLOVER, F. Heuristics for integer programming using surrogate constraints, Decision Sciences, Vol.8, pp. 156-166, 1977.

[11] GLOVER, F. McMILLAN, C. NOVICK, B. Interactive decision software and computer graphics for architetural and space planning, Annals of Operations Research, Vol.5, C.Monna, editor, pp. 557-573, 1985.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] AHO, A.V. HOPCROFT, J.E. ULLMAN, J.D. The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Company, Menlo Park - California, 1ª ed., 470 p., 1974.

[2] AHO, A.V. HOPCROFT, J.E. ULLMAN, J.D. Data Strutuctures and Algorithms, Addison-Wesley Publishing Company, Menlo Park - California, 1ª ed., 427 p., 1983.

[3] BALAS, E. An additive algorithm for solving linear programs with 0-1 variables, Operations Research, Vol.13, pp. 517-546, 1965.

[4] BALAS, E. MARTIN, C.H. Pivot and complement - a heuristic for 0-1 programming, Management Science, Vol.26(1), pp. 86-96, 1980.

[5] CÔTE, G. LAUGHTON, M.A. Large - scale mixed integer programming: Benders - type heuristics, 10th Internacional Symposium on Mathematical Programming, Montreal, August 27-31, 1979.

[12] GLOVER, F. Future paths for integer programming and links to artificial intelligence , Computers and Operations Research, Vol.13(5), pp. 533-549, 1986.

[13] GLOVER, F. McMILLAN, C. The general employee scheduling problem : an integration of management science and artificial intelligence , Computers and Operations Research, Vol.13, pp. 563-573, 1986.

[14] GLOVER, F. Tabu search methods in artificial intelligence and operations research, ORSA Artificial Intelligence Newsletter, Vol.1(2), 6 p., 1987.

[15] GLOVER, F. Tabu search, Center for Applied Artificial Intelligence - Report 88-3, 81 p., November 1988.

[16] HANSEN, P. The stepest ascent mildest descendent heuristic for combinatorial programming, presented at the Congress on Numerical Methods in Combinatorial Optimization , Capri, Italy, 1986.

[17] HANSEN, P. JAUMARD, B. Algorithms for the maximum satisfiability problem, RUCTOR Research Report-RR #43-87, Rutgers, New Brunswick, New Jersey, 1987

[18] HERTZ, A. de WERRA, D. Using tabu search techniques for graph coloring, Computing, Vol.29, pp. 345-351, 1987.

[19] HILLIER, F.S. Efficient procedures for integer linear programming with an interior , Operations Research, Vol.17, pp. 600-637, 1969.

[20] KOCHENBERGER, G.A. McCARL, B.A. WYMAN, F.P. A heuristic for general integer programming, Decision Sciences, Vol.5(1), pp. 36-46, 1974.

[21] PETERSEN, C.C. Computational experience with variants of the Balas algorithmn applied to the selection of R and D projects, Management Science, Vol.13, pp. 736-750, 1967.

[22] ROTH, R. H. An approach to solving linear discrete optimization problems, Journal of the Association for Computing Machinery, Vol.17(2), pp. 303-313, April 1970.

[23] SALKIN, H.M. Integer Programming, Addison-Wesley Publishing Company, 1ª ed., 537 p., 1975.

[24] SENJU, S. TOYODA, Y. An approach to linear programming with 0-1 variables, Management Science, Vol.15(1), pp. 303-313, 1968.

[25] TOYODA, Y. A simplified algorithm for obtaining approximate solutions to zero-one programming problem, Management Science, Vol.21(12), pp. 1417-1427, 1975.

[26] WENDELIN, C. Graph partitioning with the aid of the tabu method , Technical Report of Institute for Advanced Studies, Vienna, 1988.

[27] WYMAN, F. P. Binary programming : a decision rule for selecting optimal vs. heuristic techniques, The Computer Journal, Vol.16(2), pp. 135-140, 1973.

[28] ZANAKIS, S.H. Heuristic 0-1 linear programming: an experimental comparasion of three methods, Management Science, Vol.24(1), pp. 91-104, 1977.