UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO
MC548 – PROJETO E ANÁLISE DE ALGORITMOS II
1º Semestre de 2010
Prof. Flávio K. Miyazawa

RELATÓRIO DO PROJETO PRÁTICO

Haraldo Sérgio Albergaria Pereira Filho (003059)

1 – INTRODUÇÃO

O projeto consistiu no desenvolvimento de um programa para a resolução do "Problema da Mochila com Restrições de Conflito", que é um problema já provado ser NP-difícil, não existindo portanto um algoritmo exato de complexidade polinomial para a solução do mesmo, devendo portanto serem utilizadas outras estratégias, como por exemplo o uso de heurísticas, onde não há nenhuma garantia de se obter a solução ótima, e/ou formulando o problema em Programação Linear Inteira (PLI, que também é NP-difícil) e utilizando-se de resolvedores PL/PLI disponíveis, como por exemplo os pacotes GLPK ou CPLEX, para tentar se obter a solução ótima em tempo viável.

O programa foi executado fornecendo-se como entrada arquivos fornecidos pelo professor contendo instâncias de tamanhos variados do problema e gerando como saída arquivos com as soluções dos respectivos problemas de entrada.

2 – ESTRATÉGIA DE RESOLUÇÃO

Para a resolução do problema foi escolhida como estratégia de resolução a formulação do problema em PLI e a construção do programa ***exato*** utilizando o pacote GLPK (*GNU Linear Programming Kit*) com a implementação de uma heurística simples para ser chamada durante a execução do *branch and bound* do otimizador inteiro. Embora fosse conhecido o fato de que melhores resultados poderiam ser obtidos com o resolvedor profissional CPLEX, o GLPK foi escolhido devido estar disponível livremente e por poder ser instalado na máquina do aluno, facilitando o desenvolvimento do projeto.

3 – DEFINIÇÃO DO PROBLEMA

Sejam:

$N = \{ 0, \ldots , n-1 \}$ um conjunto de objetos, onde $n = |N|$

$M = \{ 0, \ldots , m-1 \}$ um conjunto de pessoas, onde $m = |M|$

Efetua-se um leilão, onde $i \in M$ é uma pessoa interessada em exatamente um conjunto de objetos $S_i \subseteq N$ pelo qual paga um valor p_i. Deseja-se maximizar o valor total pago ao leiloeiro, definindo $V \subseteq M$ um conjunto de pessoas $i \in V$ vencedoras do leilão, sendo que um objeto $k \in N$ não pode ser vendido para mais de uma pessoa e podem haver objetos não vendidos para ninguém.

Matematicamente, a solução do problema pode ser expressada assim:

$$\textbf{max} \{ \sum_{i \in V} p_i \ : \ \bigcap_{i \in V} S_i = \varnothing \}$$

4 – FORMULAÇÃO DO PROBLEMA EM PLI

Dada a definição do problema, a restrição de conflito que um objeto não pode ser vendido para mais de uma pessoa e o fato que podem haver objetos não vendidos para ninguém, vemos que este é um problema de empacotamento.

Assim, o problema pode ser formulado em PLI da seguinte maneira:

Maximizar: $\sum_{i \in M} p_i x_i$

Sujeito a:

$$\sum_{i \in M} s_{k,i} x_i \leq 1 \qquad \forall \ k \in N$$

$$x_i \in \{ 0, 1 \} \quad \text{e} \quad x_i = 1 \text{ se } i \in V$$
$$x_i = 0 \text{ se } i \notin V$$

$$s_{k,i} \in \{ 0, 1 \} \quad \text{e} \quad s_{k,i} = 1 \text{ se } k \in S_i$$
$$s_{k,i} = 0 \text{ se } k \notin S_i$$

5 – IMPLEMENTAÇÃO

Os programas foram implementados em linguagem C, compiláveis em Linux/gcc através de um arquivo Makefile. Embora a estratégia de resolução escolhida tenha sido a implementação do programa ***exato***, foi também implementado o programa ***heuristica***, possibilitando que as heurísticas implementadas para serem executadas pelo otimizador inteiro do GLPK pudessem ser executadas independentemente, fazendo com que pudessem ser obtidos resultados para os problemas que o programa ***exato*** não foi capaz de resolver em tempo viável. A seguir está uma breve descrição em alto nível das estruturas de dados de entrada e saída utilizadas pelos programas, da implementação do programa exato e dos algoritmos implementados nas heurísticas. Maiores detalhes sobre a implementação das funções e estruturas em C podem ser obtidas nos comentários inseridos nos aquivos dos códigos fontes e nos arquivos *headers* correspondentes.

5.1 – Estrutura de dados de entrada dos programas

Os programas lêem o arquivo de entrada passado como parâmetro e armazenam os dados lidos em uma estrutura contendo: duas variáveis inteiras **n** e **m** representando respectivamente as quantidades de objetos ***k*** no conjunto ***N*** e de pessoas ***i*** no conjunto ***M***, um vetor de variáveis inteiras **K** de ***m*** elementos representando as quantidades $\mathbf{k}_i$ de objetos que cada pessoa ***i*** deseja, um vetor de variáveis reais **P** de ***m*** elementos representando os valores p_i que cada pessoa ***i*** paga pelos $\mathbf{k}_i$ objetos desejados e uma matriz de variáveis binárias **A** de ***n*** x ***m*** elementos com as colunas representando as ***m*** pessoas ***i*** e as linhas os ***n*** objetos ***k***. A matriz **A** é o elemento mais importante dessa estrutura, sendo que para qualquer elemento $\mathbf{a}_{k,i}$ da matriz, $\mathbf{a}_{k,i}$ = 1 significa que o objeto ***k*** é desejado pelo pessoa ***i***. Ou seja, se considerarmos as colunas da matriz como vetores e as linhas como os índices dos elementos desses vetores, cada vetor corresponde à representação binária de um conjunto S_i, como exemplificada abaixo:

objetos	S_0	S_1	...	S_{i-1}	S_i	S_{i+1}	...	S_{m-2}	S_{m-1}
0	1	0	...	0	1	1	...	0	0
:	:	:	:	:	:	:	:	:	:
k-1	1	1	...	0	0	1	...	0	0
k	0	1	...	1	0	1	...	1	0
k+1	1	0	...	1	1	1	...	0	0
:	:	:	:	:	:	:	:	:	:
n-1	1	0	...	1	0	0	...	1	1

Desse modo, os valores dos elementos do vetor **K** correspondem à quantidade de objetos em cada conjunto S_i e os dos elementos do vetor **P** ao valor pago por cada conjunto, com os índices dos vetores correspondendo diretamente a cada pessoa ***i***. Esta representação foi escolhida por casar perfeitamente com a formulação em PLI do problema, pois cada elemento $a_{k,i}$ da matriz **A** corresponde exatamente a um elemento $s_{k,i}$ do programa linear inteiro e cada elemento do vetor **P** a um elemento p_i.

5.2 – Estrutura de dados de saída dos programas

Após processarem os dados de entrada os programas armazenam os resultados que serão escritos no arquivo de saída em uma estrutura de dados contendo: uma variável do tipo caractere **a** representando o algoritmo utilizado, que pode assumir os valores *'E'* ou *'e'* para o programa ***exato*** ou *'H'* para o programa ***heuristica***, uma variável inteira **t** representando o tempo em segundos gasto na resolução do problema, uma variável inteira **q** representando a quantidade de pessoas no conjunto **M**, um vetor de variáveis binárias **V** representando as pessoas vencedoras do leilão e uma variável real **s** representando o valor total pago ao leiloeiro. Os elementos mais importantes dessa estrutura são o vetor **V** e a variável real **s**, sendo que para qualquer elemento v_i de **V**, se $v_i = 1$ então a pessoa ***i*** é uma vencedora do leilão, recebendo todos os objetos do conjunto S_i e o valor p_i sendo adicionado ao valor da solução em **s**, como exemplificado abaixo:

	0	***1***	***...***	***i-1***	***i***	***i+1***	***...***	***m-2***	***m-1***
v	0	1	...	0	1	0	...	0	1
p	3	5	...	7	4	2	...	6	1

s = 5 + 4 + 1 = 10

Aqui, cada elemento v_i do vetor **V** corresponde exatamente a um elemento x_i do programa linear inteiro e **s** ao valor objetivo do problema.

5.3 – Programa *exato*

O programa ***exato***, após ler o arquivo de entrada e armazenar os dados na estrutura descrita no item 5.1, transfere esses dados para uma outra estrutura compatível com o que o GLPK aceita como entrada: três vetores **ia**, **ja** e **ar** de 1+***n********m*** elementos, onde **ia**, **ja** e **ar** armazenam respectivamente os índices **i**, os índices **j** e os valores de $\mathbf{a}_{i,j}$ na matriz de restrições do programa linear inteiro do GLPK. Os vetores têm 1+***n********m*** elementos porque o GLPK não utiliza o índice 0, sendo que os índices **i** da matriz vão de 1 a ***n*** e os índices **j** de 1 a ***m***. Assim, cada valor $s_{k,i}$ do problema original é armazenado na posição **ar**[**ia**[***k***+1] + **ja**[***i***+1]].

O GLPK recebe como entrada os vetores **ia**, **ja** e **ar**, os valores p_i, as restrições de conflito ($s_{k,i}\, x_i \leq 1$) e o tipo das variáveis estruturais (x_i binário) do problema. Além disso, vários parâmetros do GLPK são definidos através de uma estrutura própria, atribuindo valores a variáveis dessa estrutura que alteram os parâmetros. Entre esses parâmetros estão uma rotina de *callback* que será chamada pelo GLPK em alguns momentos da resolução e um ponteiro onde se pode passar informações necessárias à execução dessa rotina.

O programa executa então o algoritmo simplex do GLPK, que obtém uma solução ótima para o programa linear relaxado, que passa a ser um limitante superior para o problema. Logo em seguida, o otimizador inteiro do GLPK é executado e passa a procurar uma solução ótima inteira para o problema, parando quando a solução ótima é encontrada ou o tempo limite fornecido é atingido. Em qualquer um dos casos os resultados obtidos pelo GLPK são armazenados na estrutura de dados de saída, que é lida pelo programa e escrita no arquivo de saída.

Enquanto procura por uma solução ótima inteira, o GLPK pode chamar a rotina de *callback* recebida como parâmetro para executar uma ou mais heurísticas para obter uma solução inteira viável, que passa a ser um limitante inferior para o problema, sendo utilizada no processo de *branch&cut* do resolvedor. Essa rotina e os algoritmos implementados nas heurísticas são descritos a seguir.

5.4 – Rotina de *callback*

A rotina de callback pode ser chamada pelo otimizador inteiro do GLPK durante o processo de *branch&cut* por diversos motivos. Entre eles está a requisição de uma solução viável inteira encontrada por uma heurística, que é o único motivo tratado aqui. Assim, ao ser chamada pelo GLPK, a rotina de *callback* verifica qual o motivo de ter sido chamada e se foi por qualquer outro que não seja a execução da heuristica apenas retorna sem executar nada. Caso contrário, executa as heurísticas e devolve a solução encontrada ao GLPK, que a utilizará como um limitante inferior inteiro para o problema, podendo os ramos da árvore de enumerações a partir desse valor. Os dados originais do problema necessários para a execução das heurísticas são passados para a rotina de *callback* através

do ponteiro de informações da estrutura de configuração do GLPK juntamente com o tempo máximo de execução das heurísticas (que é definido aqui como ¼ do tempo disponível para o otimizador inteiro) e com uma variável de controle binária que verifica se as heurísticas já foram executadas, caso em que a rotina de *callback* também retorna sem executar nada. Foi necessário adicionar esse controle pois a rotina estava sendo chamada para a execução das heurísticas por muitas vezes seguidas, o que tornava o programa ***exato*** muito mais lento sem nenhum benefício, já que as heurísticas implementadas são determinísticas e retornavam sempre o mesmo resultado. Talvez essas chamadas sucessivas façam sentido se forem utilizadas heurísticas com alguma aleatoriedade, pois a cada chamada podem ser obtidos resultados diferentes e possivelmente melhores.

5.5 – Heurísticas

Foram implementadas duas variações de heurística gulosa, cada uma possuindo dois modos de execução, fazendo com que virtualmente possa ser executado um conjunto de quatro heurísticas. Foi criada uma função, que pode ser chamada tanto pela rotina de *callback* do GLPK quanto pelo programa ***heuristica***, que executa as quatro heurísticas em seqüência e devolve o melhor resultado obtido.

As duas heurísticas gulosas implementadas funcionam de maneira similar: ordenam, através do algoritmo *Quicksort,* os conjuntos $\boldsymbol{S_i}$ em ordem decrescente de um "peso", cuja escolha depende do modo em que está sendo executada, e fazem a escolha gulosa dos compradores ***i*** que farão parte da solução a partir dos conjuntos $\boldsymbol{S_i}$ de maior peso. Para isso são criados três vetores auxiliares, sendo um vetor de variáveis reais **W** com ***m*** elementos $\mathbf{w_i} = \boldsymbol{p_i} / \boldsymbol{k_i}$, um vetor de variáveis inteiras **Z** com ***m*** elementos $\mathbf{z_i}$ cujos valores representam os índices ***i*** de cada conjunto $\boldsymbol{S_i}$ que são ordenados em ordem decrescente de $\mathbf{w_i}$ ou $\boldsymbol{p_i}$, e um vetor de binários **O** com ***n*** elementos $\mathbf{o_k}$ representando os objetos que estão sendo leiloados, sendo que $\mathbf{o_k} = 1$ significa que o objeto ***k*** já foi vendido, ou seja, algum conjunto $\boldsymbol{S_i}$ a que ***k*** pertence já faz parte da solução. Assim, antes de um conjunto $\boldsymbol{S_i}$ ser adicionado à solução, os objetos ***k*** pertencentes ao conjunto são comparados um a um às posições ***k*** correspondentes no vetor **O**, sendo adicionado à solução apenas se não houver nenhum objeto ***k*** em $\boldsymbol{S_i}$ em que $\mathbf{o_k} = 1$. Essa abordagem, cuja complexidade de tempo é $O(\boldsymbol{n})$, foi escolhida por ser muito mais rápida do que a abordagem mais direta (e ingênua) que seria comparar os objetos do conjunto $\boldsymbol{S_i}$ candidato a ser adicionado à solução com cada vetor $\boldsymbol{S_i}$ já adicionado, cuja complexidade de tempo seria $O(\boldsymbol{mn})$.

A seguir temos os algoritmos em pseudocódigo e a análise da complexidade de tempo de cada heurística, onde podem ser observadas as diferenças entre elas e melhor compreendido o que foi explicado até aqui.

5.5.1 – Heurística gulosa

Esta é a heurística gulosa padrão, que apenas ordena os conjuntos S_i da forma que foi explicada anteriormente e vai fazendo a escolha gulosa, verificando cada conjunto em ordem decrescente e parando quando o último conjunto foi verificado ou quando o tempo máximo de execução foi atingido.

5.5.1.1 - Algoritmo

Heuristica Gulosa (S_i, p_i, k_i, m, ***tmax***, ***modo***)

1 Para i de 0 até m -1 faça $w_i = p_i / k_i$ e $z_i = i$
2 Caso ***modo*** é 1: Ordena vetor **Z** em ordem decrescente de w_i
2 Caso ***modo*** é 2: Ordena vetor **Z** em ordem decrescente de p_i
3 *Solução* $\leftarrow \varnothing$
4 Para i de 0 até m -1 e enquanto *tempo de execução* < ***tmax*** faça
5 Se nenhum objeto de $S_{z[i]}$ já está na *Solução*
6 Então *Solução* $\leftarrow$ *Solução* $\cup$ $S_{z[i]}$
7 Retorna *Solução*

5.5.1.2 – Análise de complexidade de tempo

1) θ (m)
2) O (m *log* m)
3) θ (1)
4-6) O (m) $\times$ O (n) = O (mn)
7) θ (1)

Complexidade total = θ (m) + O (m *log* m) + O (mn) = O (mn)

5.5.2 – Heurística gulosa estrela

Esta heurística é inspirada no algoritmo de alinhamento estrela de bioinformática, onde para se obter o melhor alinhamento múltiplo de um conjunto de sequências biológicas, escolhe-se como “pivot” a seqüência que obteve a maior pontuação na soma dos alinhamentos entre ela e as demais seqüências do conjunto. Aqui, são feitas várias iterações da heurística gulosa padrão, escolhendo em cada uma delas, em ordem decrescente de “peso”, um conjunto S_i como “pivot”, ou seja, o primeiro conjunto a ser adicionado à solução e a partir do qual começam a ser feitas as escolhas gulosas. O algoritmo pára quando o valor da

solução encontrada na iteração corrente dividido pelo valor da melhor solução encontrada até então é menor do que um fator fornecido (no caso, aqui o fator escolhido foi 0.8, ou seja, o algoritmo pára caso o valor da solução corrente seja menor que 80% do valor da melhor solução) ou quando o tempo máximo de execução é atingido, sendo que em ambos os casos a melhor solução encontrada é retornada como resposta.

5.5.2.1 – Algoritmo

Heuristica Gulosa Estrela ($\boldsymbol{S_i}$, $\boldsymbol{p_i}$, $\boldsymbol{k_i}$, $\boldsymbol{m}$, ***fator***, ***tmax***, ***modo***)

1 Para ***i*** de 0 até ***m*** -1 faça $\mathbf{w_i} = \boldsymbol{p_i} / \boldsymbol{k_i}$ e $\mathbf{z_i} = \boldsymbol{i}$
2 Caso ***modo*** é 1: Ordena vetor **Z** em ordem decrescente de $\boldsymbol{w_i}$
2 Caso ***modo*** é 2: Ordena vetor **Z** em ordem decrescente de $\boldsymbol{p_i}$
3 *Melhor Solução* $\leftarrow \varnothing$
4 Para λ de 0 até ***m*** -1 e enquanto *tempo de execução* < ***tmax*** faça
5 *Solução* $\leftarrow \varnothing$
6 *Solução* $\leftarrow$ *Solução* $\cup$ $\boldsymbol{S_{z[\lambda]}}$
7 Para ***i*** de 0 até λ -1 e enquanto *tempo de execução* < ***tmax*** faça
8 Se nenhum objeto de $\boldsymbol{S_{z[i]}}$ já está na *Solução*
9 Então *Solução* $\leftarrow$ *Solução* $\cup$ $\boldsymbol{S_i}$
10 Para ***i*** de λ +1 até ***m*** -1 e enquanto *tempo de execução* < ***tmax*** faça
11 Se nenhum objeto de $\boldsymbol{S_{z[i]}}$ já está na *Solução*
12 Então *Solução* $\leftarrow$ *Solução* $\cup$ $\boldsymbol{S_{z[i]}}$
13 Se *Solução* > *Melhor Solução*
14 Então *Melhor Solução* $\leftarrow$ *Solução*
15 Se *Solução* < ***fator*** $\times$ *Melhor Solução*
16 ou *tempo de execução* $\geq$ ***tmax***
17 Então interrompe procura por *Melhor Solução*
18 Retorna *Melhor Solução*

5.5.2.2 – Análise de complexidade de tempo

1) θ ($\boldsymbol{m}$)
2) O ($\boldsymbol{m}$ *log* $\boldsymbol{m}$)
3) θ (1)
4-17) O ($\boldsymbol{m}$) $\times$ O ($\boldsymbol{mn}$) = O ($\boldsymbol{m} \times \boldsymbol{mn}$) = O ($\boldsymbol{m^2 n}$)
18) θ (1)

Complexidade total = θ ($\boldsymbol{m}$) + O ($\boldsymbol{m}$ *log* $\boldsymbol{m}$) + O ($\boldsymbol{m^2 n}$) = O ($\boldsymbol{m^2 n}$)

6 – COMPILAÇÃO

Para a compilação dos programas foi criado um *makefile*, que compila os arquivos fonte utilizando o *gcc* do *Linux,* gerando os executáveis*.* Esse *makefile* aceita os seguintes comandos:

make exato : compila os aquivos fontes gerando o executável *exato*.

make heuristica : compila os arquivos fontes gerando o executável *heuristica*.

make [all] : compila os arquivos fontes gerando os executáveis *exato* e *heuristica*.

make clean : apaga todos os arquivos objetos e executáveis gerados.

Os seguintes comandos devem ser executados em modo *root,* antes de qualquer outro comando acima, caso o computador utilizado não tenha o pacote GLPK instalado:

make glpk-install : instala o pacote glpk-4.43 no computador.

make glpk-uninstall : desinstala o glpk e apaga todos os arquivos gerados.

7 – EXECUÇÃO

Os programas foram executados em um computador com processador *AMD Sempron 3500+* de 1,58GHz com 1,12GB de RAM e sistema operacional *Linux Fedora 7*.

Para a execução dos programas foram criados os seguintes scripts:

exato.run :

Executa o programa *exato* para os arquivos de entrada *random_*_*.in*, *frb30-15-*-*.in* e *frb35-17-*-*.in* disponíveis no diretório *instancias*, gerando no diretório *saidas/exato* os arquivos **.out* e **.scr* correspondentes, que são respectivamente o arquivo de saída contendo a solução do problema e o arquivo para onde foram direcionadas as mensagens enviadas para a saída padrão pelo programa. O tempo máximo de execução de cada arquivo foi fixado em 300 segundos (5 minutos). Os arquivos de entrada frb40-*.in a frb59-*.in não foram executados, por terem uma quantidade de dados muito grandes que travaram o computador em que foram executados em testes prévios devido à alocação de toda a memória RAM disponível.

heuristica.run :

Executa o programa *heuristica* para cada arquivo de entrada disponível no diretório *instancias*, gerando no diretório *saidas/heuristica* os arquivos **.out* e **.scr* correspondentes, que são respectivamente o arquivo de saída contendo a solução do problema e o arquivo para onde foram direcionadas as mensagens enviadas para a saída padrão pelo programa. O tempo máximo de execução de cada arquivo foi fixado em 60 segundos.

8 – RESULTADOS

Os resultados obtidos pelos programas após o término da execução estão resumidos nas tabelas a seguir. Os valores de solução com um asterisco significam que foi encontrada uma solução ótima inteira e os valores de tempo entre parêntesis significam que a execução foi interrompida devido a ter sido atingido o tempo máximo, sendo o valor da solução o melhor encontrado até aquele momento. Todos os valores de tempo estão em segundos.

Maiores detalhes sobre cada execução e os resultados correspondentes podem ser obtidos nos arquivos **.scr* e **.out* respectivamente, que estão nos diretórios *saidas/exato* e *saidas/heuristica*.

TABELA 1: $20 \leq n \leq 500$ e $20 \leq m \leq 110$

	EXATO		HEURISTICA	
PROBLEMA	**SOLUÇÃO**	**TEMPO**	**SOLUÇÃO**	**TEMPO**
random_0020_rand10	20,0*	0	20,0*	0
random_0020_size	16,0*	0	16,0*	0
random_0020_unitario	3,0*	0	3,0*	0
random_0040_rand10	22,0*	0	22,0*	0
random_0040_size	27,0*	0	27,0*	0
random_0040_unitario	4,0*	0	4,0*	0
random_0060_rand10	17,0*	0	17,0*	0
random_0060_size	41,0*	0	41,0*	0
random_0060_unitario	4,0*	0	4,0*	0
random_0080_rand10	21,0*	0	17,0	0
random_0080_size	51,0*	0	51,0*	0
random_0080_unitario	3,0*	0	3,0*	0
random_0100_rand10	26,0*	0	26,0*	0
random_0100_size	63,0*	0	63,0*	0
random_0100_unitario	4,0*	0	4,0*	0
random_0200_rand10	21,0*	0	21,0*	0
random_0200_size	135,0*	0	135,0*	0
random_0200_unitario	3,0*	0	3,0*	0
random_0300_rand10	19,0*	0	19,0*	0
random_0300_size	195,0*	0	195,0*	0
random_0300_unitario	2,0*	0	2,0*	0
random_0400_rand10	24,0*	0	24,0*	0
random_0400_size	255,0*	1	255,0*	0
random_0400_unitario	2,0*	0	2,0*	0
random_0500_rand10	29,0*	0	29,0*	0
random_0500_size	320,0*	3	320,0*	0
random_0500_unitario	3,0*	0	3,0*	0

TABELA 2: 17874 ≤ n ≤ 28143 e 450 ≤ m ≤ 595

	EXATO		HEURISTICA	
PROBLEMA	**SOLUÇÃO**	**TEMPO**	**SOLUÇÃO**	**TEMPO**
frb30-15-1-rand10	193,0	(300)	193,0	46
frb30-15-1-size	1871,0	(300)	1871,0	9
frb30-15-1-unitario	25,0	(300)	25,0	26
frb30-15-2-rand10	214,0	(300)	214,0	46
frb30-15-2-size	2011,0	(300)	2011,0	10
frb30-15-2-unitario	25,0	(300)	23,0	12
frb30-15-3-rand10	211,0	(300)	211,0	34
frb30-15-3-size	1875,0	(300)	1875,0	19
frb30-15-3-unitario	24,0	(300)	24,0	38
frb30-15-4-rand10	193,0	(300)	193,0	55
frb30-15-4-size	1915,0	(300)	1915,0	17
frb30-15-4-unitario	25,0	(300)	25,0	5
frb30-15-5-rand10	211,0	(300)	211,0	25
frb30-15-5-size	1926,0	(300)	1926,0	7
frb30-15-5-unitario	24,0	(300)	24,0	20
frb35-17-1-rand10	229,0	(300)	229,0	(60)
frb35-17-1-size	2530,0	(300)	2515,0	23
frb35-17-1-unitario	29,0	(300)	28,0	(60)
frb35-17-2-rand10	227,0	(300)	227,0	(60)
frb35-17-2-size	2560,0	(301)	2560,0	17
frb35-17-2-unitario	29,0	(300)	29,0	(60)
frb35-17-3-rand10	222,0	(300)	222,0	(60)
frb35-17-3-size	2594,0	(300)	2594,0	43
frb35-17-3-unitario	29,0	(300)	28,0	(60)
frb35-17-4-rand10	228,0	(300)	228,0	(60)
frb35-17-4-size	2429,0	(300)	2429,0	6
frb35-17-4-unitario	28,0	(300)	28,0	(60)
frb35-17-5-rand10	218,0	(300)	218,0	(60)
frb35-17-5-size	2699,0	(300)	2699,0	59
frb35-17-5-unitario	28,0	(300)	28,0	(60)

TABELA 3: $41095 \leq n \leq 81068$ e $760 \leq m \leq 1150$

	EXATO		HEURISTICA	
PROBLEMA	**SOLUÇÃO**	**TEMPO**	**SOLUÇÃO**	**TEMPO**
frb40-19-1-rand10	X	X	267,0	(60)
frb40-19-1-size	X	X	3391,0	(60)
frb40-19-1-unitario	X	X	32,0	(60)
frb40-19-2-rand10	X	X	259,0	(60)
frb40-19-2-size	X	X	3647,0	(60)
frb40-19-2-unitario	X	X	33,0	(60)
frb40-19-3-rand10	X	X	248,0	(60)
frb40-19-3-size	X	X	3385,0	(60)
frb40-19-3-unitario	X	X	31,0	(60)
frb40-19-4-rand10	X	X	254,0	(60)
frb40-19-4-size	X	X	3424,0	(60)
frb40-19-4-unitario	X	X	31,0	(60)
frb40-19-5-rand10	X	X	263,0	(60)
frb40-19-5-size	X	X	3450,0	(60)
frb40-19-5-unitario	X	X	31,0	(60)
frb45-21-1-rand10	X	X	289,0	(60)
frb45-21-1-size	X	X	4318,0	(60)
frb45-21-1-unitario	X	X	34,0	(60)
frb45-21-2-rand10	X	X	281,0	(61)
frb45-21-2-size	X	X	4399,0	(60)
frb45-21-2-unitario	X	X	34,0	(60)
frb45-21-3-rand10	X	X	276,0	(60)
frb45-21-3-size	X	X	4349,0	(60)
frb45-21-3-unitario	X	X	34,0	(60)
frb45-21-4-rand10	X	X	285,0	(60)
frb45-21-4-size	X	X	4322,0	(60)
frb45-21-4-unitario	X	X	35,0	(60)
frb45-21-5-rand10	X	X	284,0	(60)
frb45-21-5-size	X	X	4284,0	(60)
frb45-21-5-unitario	X	X	35,0	(60)
frb50-23-1-rand10	X	X	316,0	(60)
frb50-23-1-size	X	X	5456,0	(60)
frb50-23-1-unitario	X	X	38,0	(60)
frb50-23-2-rand10	X	X	339,0	(61)
frb50-23-2-size	X	X	5258,0	(61)
frb50-23-2-unitario	X	X	39,0	(60)
frb50-23-3-rand10	X	X	310,0	(61)
frb50-23-3-size	X	X	5342,0	(61)
frb50-23-3-unitario	X	X	38,0	(61)
frb50-23-4-rand10	X	X	318,0	(60)
frb50-23-4-size	X	X	5395,0	(61)
frb50-23-4-unitario	X	X	38,0	(61)
frb50-23-5-rand10	X	X	310,0	(60)
frb50-23-5-size	X	X	5421,0	(61)
frb50-23-5-unitario	X	X	39,0	(62)

TABELA 4: 94127 ≤ n ≤ 127011 e 1272 ≤ m ≤ 1534

	EXATO		HEURISTICA	
PROBLEMA	**SOLUÇÃO**	**TEMPO**	**SOLUÇÃO**	**TEMPO**
frb53-24-1-rand10	X	X	342,0	(61)
frb53-24-1-size	X	X	5739,0	(62)
frb53-24-1-unitario	X	X	40,0	(61)
frb53-24-2-rand10	X	X	340,0	(61)
frb53-24-2-size	X	X	6024,0	(61)
frb53-24-2-unitario	X	X	40,0	(60)
frb53-24-3-rand10	X	X	331,0	(61)
frb53-24-3-size	X	X	5762,0	(60)
frb53-24-3-unitario	X	X	42,0	(61)
frb53-24-4-rand10	X	X	325,0	(61)
frb53-24-4-size	X	X	5857,0	(62)
frb53-24-4-unitario	X	X	40,0	(62)
frb53-24-5-rand10	X	X	335,0	(60)
frb53-24-5-size	X	X	5932,0	(60)
frb53-24-5-unitario	X	X	40,0	(61)
frb56-25-1-rand10	X	X	345,0	(60)
frb56-25-1-size	X	X	6369,0	(62)
frb56-25-1-unitario	X	X	41,0	(62)
frb56-25-2-rand10	X	X	363,0	(62)
frb56-25-2-size	X	X	6315,0	(60)
frb56-25-2-unitario	X	X	43,0	(63)
frb56-25-3-rand10	X	X	337,0	(61)
frb56-25-3-size	X	X	6510,0	(62)
frb56-25-3-unitario	X	X	41,0	(60)
frb56-25-4-rand10	X	X	354,0	(60)
frb56-25-4-size	X	X	6844,0	(60)
frb56-25-4-unitario	X	X	41,0	(61)
frb56-25-5-rand10	X	X	344,0	(62)
frb56-25-5-size	X	X	6507,0	(62)
frb56-25-5-unitario	X	X	41,0	(63)
frb59-26-1-rand10	X	X	382,0	(62)
frb59-26-1-size	X	X	7451,0	(60)
frb59-26-1-unitario	X	X	44,0	(63)
frb59-26-2-rand10	X	X	391,0	(60)
frb59-26-2-size	X	X	6845,0	(61)
frb59-26-2-unitario	X	X	43,0	(61)
frb59-26-3-rand10	X	X	377,0	(62)
frb59-26-3-size	X	X	7106,0	(63)
frb59-26-3-unitario	X	X	44,0	(62)
frb59-26-4-rand10	X	X	364,0	(63)
frb59-26-4-size	X	X	7389,0	(62)
frb59-26-4-unitario	X	X	45,0	(61)
frb59-26-5-rand10	X	X	365,0	(62)
frb59-26-5-size	X	X	7312,0	(60)
frb59-26-5-unitario	X	X	42,0	(62)