Relatório do trabalho prático – MC548 2011s1

Instituto de Computação – Unicamp – Prof. Cid Souza

27/06/2011

**1 - Integrantes do grupo:**

Alberto Arruda de Oliveira RA 093311

Luiz Claudio C. de Carvalho RA 800578

**2 - parte 1 do trabalho:**

Exercício [gt54]:

Variáveis utilizadas: **x[i,j]**, binária, com (i,j) pertencente a A, tal que:

**x[i,j] = 1** se (i,j) pertence ao caminho;

**x[i,j] = 0** caso contrário;

Restrições:

**{u in V: u != s and u != t}: sum{i in V: (i,u) in A}x[i,u] = sum{j in V: (u,j) in A}x[u,j];**

Garante que o número de arcos no caminho que saem de um vértice qualquer (exceto t e s) seja igual ao número de arcos que entram nele, garantindo então que seja um caminho.

**{(i,j) in C}: sum{u in V: (i,u) in A}x[i,u] + sum{u in V: (u,j) in A}x[u,j] <= 1;**Garante que o caminho tenha no máximo um vértice de cada par de vértices em C.

**sum{j in V: (s,j) in A}x[s,j] = 1;**

Garante que saia exatamente um arco do vértice s, tornando-o, de fato, fonte.

**sum{i in V: (i,t) in A}x[i,t] = 1;**

Garante que entre exatamente um arco no vértice t, tornando-o, de fato, destino.

Função Objetivo: **minimizar z = sum{(i,j) in A}x[i,j]\*c[i,j]**

Onde c[i,j] é o custo do arco.

Exercício [ss2]:

Variáveis utilizadas: **x[i]**, binária, com i pertencente a A, tal que:

**x[i] = 1** se a tarefa i estoura o deadline;

**x[i] = 0** caso contrário;

**sigma[i]**, inteira e maior ou igual a 0;

Indica o instante de início da tarefa i;

**prec[i,j]**, com i e j pertencentes a A, i diferente de j e (i,j) não pertencente a S, binária

**prec[i,j] = 1** se a tarefa i precede a tarefa j;

**prec[j,i] = 0** se a tarefa j precede a tarefa i;

Restrições:

**{(i,j) in S}: sigma[i] + t[i] <= sigma[j];**Garante a precedência de pares de tarefas que estejam em S.

**{i in A, j in A: i != j and (i,j) not in S and (j,i) not in S}: prec[i,j] + prec[j,i] = 1;  
{i in A, j in A: (i,j) not in S and (j,i) not in S and i != j}: sigma[i]+t[i]<=sigma[j]+(1–prec[i,j])\*M;  
{i in A, j in A: (i,j) not in S and (j,i) not in S and i != j}: sigma[j] + t[j] <= sigma[i] + prec[i,j]\*M;**Garantem a ordem de precedência das tarefas que não estão em S e a não execução de duas tarefas simultaneamente.

**{i in A}: d[i] - sigma[i] - t[i] <= (1 – x[i])\*M;  
{i in A}: d[i] - sigma[i] - t[i] >= -x[i]\*M;**

Garantem que se a tarefa i estoura o deadline, o seu instante de início sigma[i] seja maior que d[i].

Função Objetivo: **minimizar z = sum{i in 1..n}x[i];**

Minimiza o número de tarefas que termina depois do deadline

Exercício [gt10]:

Variáveis utilizadas: **x[i,j]**, binária, com (i,j) pertencente à E, tal que:

**x[i,j] = 1** se (i,j) pertence ao emparelhamento;

**x[i,j] = 0** caso contrário.

Restrições:

**{u in V}: sum{i in V: (i,u) in E}x[i,u] + sum{j in V: (u,j) in E}x[u,j] <= 1;**Garante que as arestas tomadas formem um emparelhamento, ou seja, pra cada vértice, o máximo de arestas incidentes nele que podem pertencer ao emparelhamento é 1.

**{(i,j) in E}:x[i,j] + sum{u in V: (i,u) in E and u != j}x[i,u] +**

**sum{u in V: (u,i) in E}x[u,i] +**

**sum{u in V: (j,u) in E}x[j,u] +**

**sum{u in V: (u,j) in E and u != i}x[u,j] >= 1;**

Garante que o emparelhamento seja maximal, ou seja, para todas as arestas do grafo devemos ter que ela ou pelo menos uma aresta adjacente a ela esteja no emparelhamento.

Função Objetivo: **minimizar z = sum{(i,j) in E} x[i,j];**

Minimiza o total de arestas presentes no emparelhamento.

Exercício [mn27]:

Variáveis utilizadas: **cor[i]**, binária, com 1 <= i <= p, tal que:

**cor[i] = 1** se a cor de índice i foi usada;

**cor[i] = 0** caso contrário;

**x[i,j]**, binária, com i pertencente a V e 1<= j <= p, tal que:

**x[i,j] = 1** se o vértice i foi colorido com a cor j;

**x[i,j] = 0** caso contrário;

Restrições:

**{i in V}: sum{c in 1..p}x[i,c] = 1;**Garante que todos os vértices sejam coloridos com exatamente uma cor.

**{(i,j) in E, c in 1..p}: x[i,c] + x[j,c] <= cor[c];**

Garante que vértices adjacentes não tenham mesma cor, como também que se uma cor é usada, ela é marcada como usada.

Função Objetivo: **minimizar z = sum{c in 1..p}cor[c];**

Minimiza o total de cores usadas;

Exercício [nd32]:

Variáveis utilizadas: **flow[i,j]**, inteira, maior ou igual a 0, indica a quantidade de fluxo que

passa no arco (i,j);

**x[i,j]**, binária, com (i,j) pertencente a A, tal que:

**x[i,j] = 1** se passa fluxo no arco (i,j);

**x[i,j] = 0** caso contrário.

Restrições:

**{(i,j) in A}:flow[i,j] <= c[i,j]\*x[i,j];**Garante que o fluxo em uma aresta não exceda sua capacidade;

**{v in V: v != s and v != t}: sum{i in V: (i,v) in A}flow[i,v] =**

**sum{j in V: (v,j) in A}flow[v,j];**

Garante que haja conservação de fluxo em todos os vértices (exceto s e t) do grafo.

**sum{i in V: i != t and (i,t) in A}flow[i,t] = R;**

Garante que seja atendida a demanda em t.

Função Objetivo: **minimizar z = sum{(i,j) in A}x[i,j]\*p[i,j];**

Minimiza o custo referente as arestas usadas. Note que o custo não é unitário, e sim um custo de utilização absoluto, portanto a necessidade da variável que diz se fluxo passa em uma aresta ou não.

Parte 1 - Resultados:

|  |  |  |  |
| --- | --- | --- | --- |
| **Exercício** | **Instância** | **Função Objetivo** | **F.O.2** |
| [gt54] | 1 | 6 | 6 |
| [gt54] | 2 | 26 | 26 |
| [gt54] | 3 | 20 | infeasible |
| [ss2] | 1 | 1 | 1 |
| [ss2] | 2 | 6 | 6 |
| [ss2] | 3 | 17 | 17 |
| [gt10] | 1 | 1 | 1 |
| [gt10] | 2 | ?? | 8 |
| [gt10] | 3 | ?? | ?? (1h) |
| [mn27] | 1 | 3 | 3 |
| [mn27] | 2 | 7 | ??(1h) |
| [mn27] | 3 | ? | n.t. |
| [nd32] | 1 | 12 | 12 |
| [nd32] | 2 | 55 | 55 |
| [nd32] | 3 | 113 | 113 |

**3 - parte 2 do trabalho:**

Estruturas de dados utilizadas:

Foi construída uma estrutura para cada um dos elementos básicos do problema, a saber, os **satélites** e as **shards**:

shard

int posH; //Posição Horizontal

int posV; //Posição Vertical

float rShard; //Ganho de Armazenagem

float cH; //Custo de armazenagem pelo satélite em

// rota horizontal

float cV; //Custo de armazenagem pelo satélite em

// rota vertical

int lidaPor;

bool lida;

satelite

int ns; //Numero do satélite

float memTotal; //Memória total do satélite

float memRestante; //Memória restante do satélite

Foi criados também dois vetores de inteiros, currentSolution e bestSolution, para representar soluções do problema, onde seus índices coincidem com os do vetor de shards (ver abaixo) e o valor correspondente corresponde ao do úmero do satélite que a leu, nesta solução.

\*\*\*Verificar a utilidade de se criar, na estruct do satélite, um vetor com apontadores para as shards que leu (ou pode ler).\*\*\*

Leitura de dados:

Foi criada uma função específica para este fim. A mesma armazena em vetores das estruturas acima os satélites e as shards fornecidas na entrada. Retorna valor total de rShard, ou seja, da recompensa máxima que se pode extrair da instância. Os satélites são numerados de 1 a 2n, sendo que os satélites de rota vertical recebem os números (satélite.ns) de *n+1* a *2n*. Isso permite que estes sejam armazenados em um único vetor, facilitando sua ordenação. Para facilitar as comparações, a coordenada vertical de cada shard recebeu o valor *y+n*, ou seja, ao valor da entrada foi acrescida a quantidade de satélites.

Após a leitura, estes vetores foram **ordenados** da seguinte forma:

* Satélites: ordem crescente de capacidade de memória;
* Shards: ordem decrescente da relação benefício/custo, tendo sido utilizado o menor dos dois custos de leitura (horizontal ou vertical);

Pré-processamento:

Verificou-se pelo exame das instâncias que existem shards que não podem ser lidas por nenhum dos dois satélites que a sobrevoam. Verificado o caso, as shards correspondentes são eliminadas do problema. Da mesma maneira, verificou-se que existem satélites que não são capazes de ler quaisquer das shards que sobrevoam, seja por que sua memória é inferior aos custos de ambas as shards sobrevoadas ou porque não existe nenhum alvo em sua trajetória. Estes satélites também foram eliminados, e a coordenada correspondente nas shards sobrevoadas por eles foram trocadas pelo negativo de seu valor.

Heurística básica:

O problema foi tratado como uma sequência de problemas da mochila, sendo alocadas a cada satélite as shards viáveis e ainda não lidas que maximizam a recompensa capturada por cada um. Para a solução de cada subproblema de preenchimento de um satélite usamos a heurística ingênua de se alocar os melhores R/C, ou seja, as shards são alocadas na sequência em que foram ordenadas após sua leitura. Para minimizar a chance de que um satélite de menor capacidade não fosse utilizado, estes são preenchidos primeiro, ou seja, também de acordo com sua ordenação inicial. A complexidade desta heurística básica é O(n.m), sendo n e m o número de satélites e shards, respectivamente.

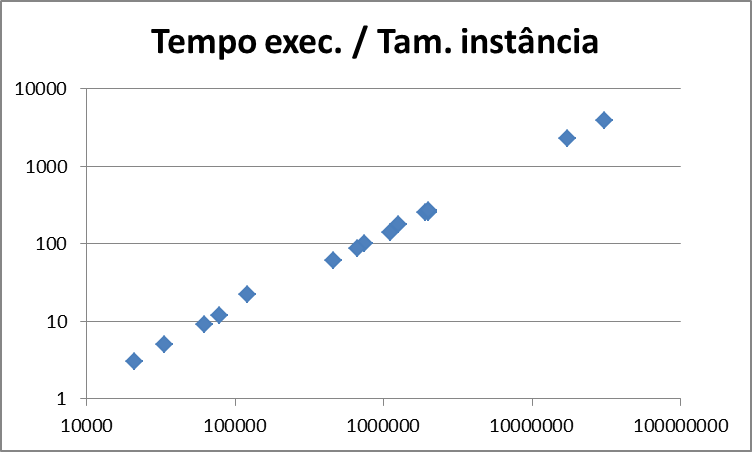
Os resultados da aplicação desta heurística básica são tabulados abaixo.

Tabela.3.1 – resultados da heurística básica

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Instância** | **Satélites** | **Shards** | **R. Máximo** | **Capturado** | **%** | **Tempo exec.\*** |
| Small0 | 80 | 264 | 13.666 | 8.450 | **62** | 3 |
| Small1 | 80 | 1.520 | 76.022 | 53571 | **70** | 22 |
| Small2 | 80 | 774 | 39.212 | 27.166 | **69** | 9 |
| Small3 | 80 | 420 | 21.307 | 14.106 | **66** | 5 |
| Small4 | 80 | 975 | 49.391 | 30.763 | **62** | 12 |
| Med0 | 200 | 9.508 | 4.718.409 | 3.185.460 | **68** | 253 |
| Med1 | 200 | 3.719 | 1.845.683 | 1.290.119 | **70** | 100 |
| Med2 | 200 | 10.000 | 5.002.484 | 3.423.897 | **68** | 255 |
| Med3 | 200 | 5.528 | 2.804.919 | 1.833.207 | **65** | 141 |
| Med4 | 200 | 10.000 | 5.021.152 | 3.524.435 | **70** | 267 |
| Med5 | 200 | 6.341 | 3.194.956 | 2.257.615 | **71** | 175 |
| Med6 | 200 | 2.297 | 1.139.797 | 715.828 | **63** | 61 |
| Med7 | 200 | 3.330 | 1.690.257 | 1.115.662 | **66** | 87 |
| Big0 | 600 | 29.069 | 14.510.868 | 9.807.963 | **68** | 2262 |
| Big1 | 600 | 50.913 | 25.453.577 | 18.316.737 | **72** | 3870 |

\* variação em clock() durante a rotina de alocação;

Fig. 3.1 tempos de execução – heurística básica\*



\* escala di-logaritmica. Tempo de execução em ticks versus dimensão da entrada

(shards x satélites).

O exame da alocação produzida por esta heurística mostrou que os satélites de maior capacidade, preenchidos por último, frequentemente ficavam bastante ociosos. Esta constatação inspirou o aprimoramento introduzido, descrito a seguir.

(\*\*\* Caso ainda haja memória disponível em algum dos satélites e existam shards que não foram lidas\*\*\*), os satélites são novamente percorridos, desta vez no sentido inverso da ordenação (dos maiores para os menores). Para cada um deles, são percorridas novamente todas as shards, testando-se as seguintes condições:

* Pode ser lida pelo satélite atual;
* Não está alocada ou foi alocada anteriormente a um satélite de menor capacidade;

Preenchidas estas condições, a shard é alocada ao satélite atual e a memória consumida por ela é devolvida ao satélite onde se encontrava anteriormente alocada. (\*\*\* Se este procedimento já não reduz a zero a quantidade de shards não lidas\*\*\*), as duas rotinas (*heurística básica* e *deslocamento para os maiores*) são executadas sequencialmente, enquanto houver melhorias ou até que todas as shards tenham sido lidas.

No que diz respeito à sequência em que as shards são percorridas em cada laço do deslocamento para os satélites maiores, foram feitas três tentativas: o percurso na ordem original (recompensa/custo decrescente), a ordem inversa e a escolha aleatória de se uma shard apropriada para deslocamento é efetivamente deslocada ou não. Neste último caso, foi considerado um número máximo de iterações sem melhoria para a interrupção da rotina. Os resultados de cada variante são sintetizados na tabela abaixo (apenas para as instâncias em que a heurística básica não foi 100% eficiente). A memória total restante nos satélites tabulada é apenas uma indicação e não uma comprovação de que foi atingida uma solução ótima para a instância. (\*\*\* Os testes efetuados com substituição da heurística básica por uma invocação ao solver glpk para a alocação de shards aos satélites produziram os limitantes superiores indicados na última coluna\*\*\*).

O código entregue no trabalho corresponde ao da variação \*\*\*N\*\*\*

Tabela 3.2 – Aprimoramentos comparados

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Heurística** | **Básica** | | | **Variação 1** | | | **Variação 2** | | | **Variação 3** | | | **LS** |
| **Instância** | **NC** | **MR** | **TE** | **NC** | **MR** | **TE** | **NC** | **MR** | **TE** | **NC** | **MR** | **TE** | **NC** |
| Small2 | 31 |  | 0 |  |  |  |  |  |  |  |  |  |  |
| Med0 | 1.114 |  | 37 |  |  |  |  |  |  |  |  |  |  |
| Med1 | 360 |  | 14 |  |  |  |  |  |  |  |  |  |  |
| Med2 | 130 |  | 16 |  |  |  |  |  |  |  |  |  |  |
| Med4 | 96 |  | 13 |  |  |  |  |  |  |  |  |  |  |
| Med5 | 184 |  | 13 |  |  |  |  |  |  |  |  |  |  |
| Med6 | 6 |  | 13 |  |  |  |  |  |  |  |  |  |  |
| Med7 | 236 |  | 23 |  |  |  |  |  |  |  |  |  |  |
| Big0 | 27 |  | 724 |  |  |  |  |  |  |  |  |  |  |
| Big1 | 5.743 |  | 1.566 |  |  |  |  |  |  |  |  |  |  |

NC – Recompensa não capturada

MR – Memória não utilizada dos satélites

TE – tempo de execução

LS – limitante superior obtido com solver glpk

A complexidade de pior caso das variantes 1 e 2 é potencialmente exponencial, dependendo do número de iterações que forem executadas até que se chegue à solução ótima ou que não se obtenham melhorias na solução. A da variação 3 depende do tamanho da entrada e do número máximo de iterações que forem permitidas. Na prática foi observado que o número máximo de iterações realizadas foi de (\*\*\*N\*\*\*). O gráfico abaixo os tempos de execução em função do tamanho da entrada de cada uma das instâncias.