A01 - AED

Generalized weighted job selection problem

Trabalho realizado por:

Henrique Sousa

NºMec 98324, P4

Índice

Introdução

Dadas T tarefas de programação, cada uma com uma data de início, uma data de fim e um lucro e dados P programadores, o problema de seleção de trabalho ponderado generalizado pede o melhor subconjunto de tarefas de programação, tal que: a soma dos lucros das tarefas pertencentes ao subconjunto é superior, cada tarefa de programação é feita por um único programador e cada programador não pode trabalhar em mais de uma tarefa ao mesmo tempo.

De acordo com as regras fornecidas acima, um programador não pode interromper uma tarefa de programação para fazer outra. Uma vez que o programador está comprometido com uma tarefa, este está ocupado até que a mesma termine.

Estrutura do Código

Para simplificar o entendimento do algoritmo fiz uma função (recursiva) separada do código principal que é chamada na função “solve” dada, previamente, pelos docentes:

```
static void solve(problem_t  *problem)
{
  FILE *fp;
  int i;

  //
  // open log file
  //
  (void)mkdir(problem->dir_name,S_IRUSR | S_IWUSR | S_IXUSR);
  fp = fopen(problem->file_name,"w");
  if(fp == NULL)
  {
    fprintf(stderr,"Unable to create file %s (maybe it already exists? If so,
delete it!)\n",problem->file_name);
    exit(1);
  }
  //
  // solve
  //
  problem->cpu_time = cpu_time();
 int jobs[problem->T], bestJobs[problem->T];
 problem->viableCount = 0;
 problem->nCases = 0;
 FILE *f = fopen("table098324.txt", "a");
 solution(jobs, 0, bestJobs, problem);
 fprintf(f, "%d\t%d\t%d\t%d\t%d\t", problem->T, problem->P, problem->nCases,
problem->viableCount, problem->bestTotalProfit);
 for (int j = 0; j < problem->T; j++)
  fprintf(f, "%d", bestJobs[j]);
 fprintf(f, "\n");
 fclose(f);
  problem->cpu_time = cpu_time() - problem->cpu_time;
  //
  // save solution data
  //
... }
```

Nesta secção do código são inicializadas algumas variáveis e um ficheiro é aberto para guardar os valores obtidos a partir da função

“solution”. Esta função tem as seguintes linhas de código comentadas com a sua explicação:

```
void solution(int jobs[], int i, int bestJobs[], problem_t *problem)
{
  if (i == problem->T) {
    problem->nCases++;

    // visit all
    // assigned to phase
    for (int j = 0; j < problem->P; j++)
      {
        problem->busy[j] = -1;
      }
    int assignedCount = 0;
    for (int j = 0; j < problem->T; j++)
    {
        if (jobs[j] == 0)
        {
          problem->task[j].assigned_to = -1;
        }
        else if (jobs[j] == 1)
        {
          for (int p = 0; p < problem->P; p++)
          {
            if (problem->busy[p] < problem->task[j].starting_date)
            {
              problem->task[j].assigned_to = p;
              problem->busy[p] = problem->task[j].ending_date;
              assignedCount += 1;
              break;
            }
          }
        }
      }
      int sum = 0;
      assignedCount = 0;
      for (int j = 0; j < problem->T; j++)
      {
        sum += jobs[j]; //sum will give us the number of jobs to do
        if (problem->task[j].assigned_to >= 0)
          assignedCount++; //assigned count will give us the number of tasks
with programmers
      }
      if (assignedCount == sum) // the job is viable
      {
        problem->viableCount++;
```

```
      //profit calculation
        problem->total_profit = 0;
        for (int j = 0; j < problem->T; j++)
        {
          if (problem->task[j].assigned_to >= 0)
            problem->total_profit += problem->task[j].profit;
        }
      //end of profit calculation

      //check if new profit > than the actual best Profit
        if (problem->total_profit > problem->bestTotalProfit)
        {
          problem->bestTotalProfit = problem->total_profit;
          //change new best profit combination
          for (int k = 0; k < problem->T; k++)
            bestJobs[k] = jobs[k];

        }
      //end of checking the best new profit
      }
    // end of assigned to phase
    return;
  }

  jobs[i] = 0;
  solution(jobs, i + 1, bestJobs, problem);


  // And then assign "1" at ith position
  // and try for all other permutations
  // for remaining positions

  jobs[i] = 1;
  solution(jobs, i + 1, bestJobs, problem);
}
```

Primeiro Algoritmo

A função “solution” mostrada acima foi o primeiro método que eu usei para resolver o problema principal. Esta função é chamada recursivamente até a variável “i” ser igual ao número total de tasks. “i” é o index que indica a posição no array jobs[]. Este array tem uma dimensão igual ao número total de tasks e cada index representa uma task e é apenas composto por uns e zeros. Se jobs[i] é igual a “1” significa que a task será feita, caso contrário, estará definido a “0” e não será realizada. A função é chamada recursivamente para criar todas as combinações binárias, “i” é incrementado todas as vezes que a função é chamada, logo quando “i” é igual ao número total de tasks significa que cada task já foi definida ou a 0 ou 1 e então já podemos resolver o problema para este set específico de tasks.

“solution” é sempre chamada com quatro argumentos: "`int jobs[]`" é um array de inteiros que tem dimensão "problem->T" cada um representando uma task como foi dito anteriormente; "`int i`" é a variável que representa uma posição do array jobs[]; "`int bestJobs[]`" é um array de inteiros que tem a mesma dimensão que jobs[] e que cada posição representa uma task. A diferença entre este array e o array jobs[] é que bestJobs[] guarda o set de tasks do qual resulta o maior lucro até àquele momento; "`problem_t *problem`" necessita de ser adicionado à função para poder aceder às variáveis associadas à struct problem_t.

Melhores Lucros Obtidos

Depois de rodar “job_selection_do_all.bash” para {1...36} tarefas e {1...10} programadores para o número mecanográfico 98324. Obtive os seguintes resultados:

Linhas → Número de Tarefas

Colunas → Número de Programadores

Nmec 98324	1	2	3	4	5	6	7	8	9	10
1	4018	-	-	-	-	-	-	-	-	-
2	4958	3875	-	-	-	-	-	-	-	-
3	2875	2634	4517	-	-	-	-	-	-	-
4	4771	6071	7058	10226	-	-	-	-	-	-
5	3626	7630	7088	9983	10528	-	-	-	-	-
6	4100	8443	12035	10331	11795	9791	-	-	-	-
7	9889	6004	11578	9516	9170	14163	16936	-	-	-
8	7753	8355	7495	12636	12773	19065	16852	14370	-	-
9	11276	9263	8820	13348	20895	12954	13862	16473	21947	-
10	18773	12222	14472	14267	12811	17058	19911	17707	21958	28276
11	19939	13171	14041	15556	18559	19367	11720	17182	19337	25165
12	13740	14195	13193	15139	18312	21576	21252	22657	25200	21949
13	16129	9924	20236	19475	17333	17998	20500	20819	26005	26165
14	13187	15720	21990	16296	17365	26589	24725	27294	29226	30266
15	19616	16906	18780	20842	15621	20716	25720	28764	23895	28229
16	18246	17888	18231	16714	31576	19371	24270	31656	37845	25888
17	29416	25110	18832	24459	27604	22487	30485	31082	27496	25868
18	38215	24939	25773	26389	23971	23383	26539	27564	30138	30078

19	30117	24167	27660	26392	28602	25363	26657	28400	33769	30568
20	23617	36597	27672	21561	23809	27458	31608	25943	25163	37617
21	23870	31382	32060	26823	28038	25030	26445	31694	30359	34948
22	24130	31358	30352	36391	25611	25028	24728	31584	26602	39154
23	20814	39677	43207	38674	27877	34285	31934	31698	35435	31679
24	40687	30337	36206	38088	26699	38406	37833	27055	32816	29560
25	52643	30361	26762	37499	35819	32398	36293	39602	34310	38090
26	30742	43764	40906	42004	39712	39317	34944	39360	37918	36857
27	37757	31216	38810	40590	37187	37633	30287	38734	36579	39166
28	32437	36272	45002	37461	38691	39418	36970	32532	41687	41476
29	47245	36831	44922	34613	51126	46038	46140	36454	37932	42941
30	36741	41011	42384	47985	51329	47955	38097	45837	41398	42820
31	33793	39237	52470	47737	43335	49555	44164	40808	38837	39029
32	42307	44666	41847	50484	37878	46567	39889	42146	43814	46911
33	34304	47770	50803	52204	55356	48753	54161	46456	46058	42553
34	49615	45262	53441	46089	57160	51150	54857	43925	47333	41142
35	59156	53119	40571	53295	51159	58376	43489	60808	36319	49876
36	52133	42838	55184	58511	46681	52696	51056	50486	51420	47776

Gráficos

Como obtive os gráficos

Para obter os resultados necessários para criar e analisar os gráficos usei um script em bash que recolhe as informações necessárias e guarda-as num ficheiro para, mais tarde, ser lido pelo matlab para criar o gráfico. O script “extractData.bash” tem o seguinte conteúdo:

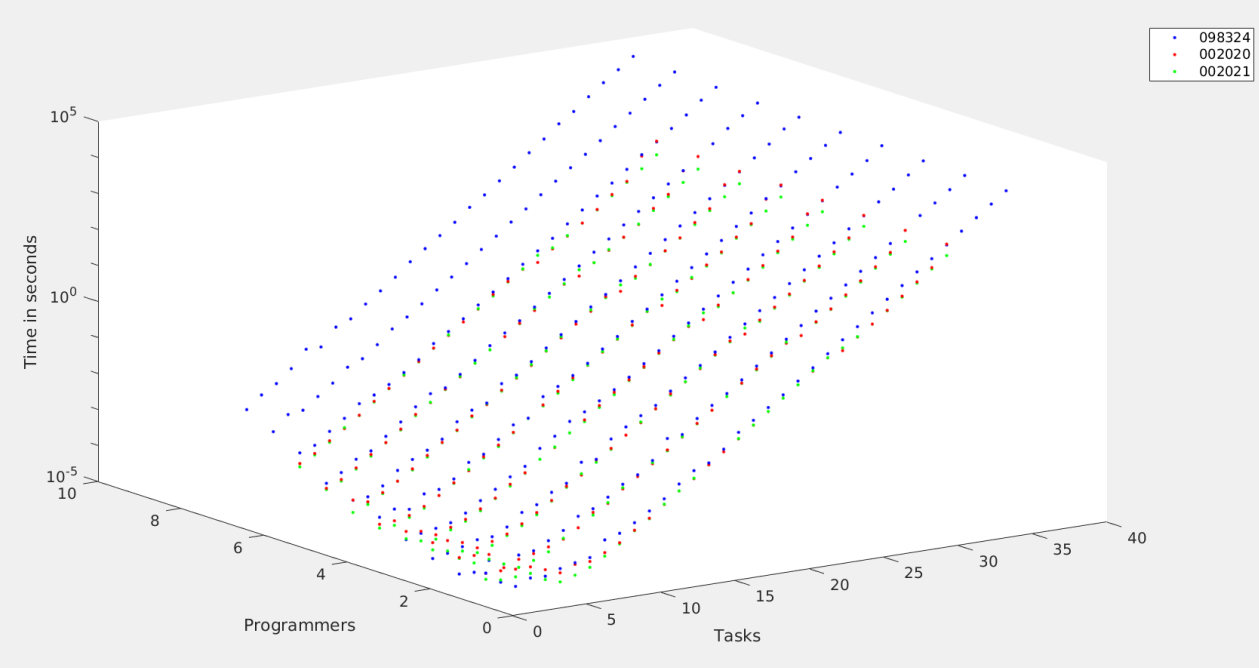
```
#! /bin/bash

dir=$1
file=data$dir.txt
if [ -e $file ]; then
  rm $file
fi
touch $file
for jobSeletion in $(ls $dir); do
    T=$( grep '^T' $dir/$jobSeletion | tr -dc '0-9' )
    P=$( grep '^P' $dir/$jobSeletion | tr -dc '0-9' )
    solutionTime=$( grep "Solution time =" $dir/$jobSeletion| cut -d ' ' -f
'4-' )
    printf "%d \t %d \t %s \n" "$T" "$P" "$solutionTime" >> $file
done
```

Este script cria o ficheiro “data(NºMec).txt”. Para ter mais valores no gráfico rodei o programa, mas desta vez apenas até 32 tarefas com 8 programadores para os números 2020 e 2021. Rodando este script para os três números foram criados três ficheiros: data098324.txt; data002020.txt; data002021.txt.

Com estes ficheiros é possível agora criar os gráficos com o seguinte programa de Matlab “graphJobSelection.m”:

```
nMecs= ["098324", "002020", "002021"];
color = "blue";
format long
for i = 1 : length(nMecs)
    file = strcat('data',nMecs(i),'.txt');
    data = load(file);
    T = data(:,1);
    P = data(:,2);
```

```
    time = data(:,3);
    plot3(T, P, time, '.', "color", color, "DisplayName", nMecs(i));
    hold on
    if color == "blue"
        color = "red";
    else
        color = "green";
    end
end
xlabel('Tasks')
ylabel('Programmers')
zlabel('Time in seconds')
set(gca,'zscale','log');
legend show
hold off
```



Apenas rodei os números “2020” e “2021” para até 32 tarefas com 8 programadores. O número “principal” 98324 tentei rodar até ao máximo de 50 tarefas com 10 programadores. Com isto, o valor máximo de tarefas que consegui resolver com o primeiro algoritmo para 10 programadores foram 36 tarefas.

Maior Número de Tarefas

Correndo o primeiro algoritmo até um máximo de 50 tarefas para um máximo de 10 programadores percebi que a partir de 37 tarefas já não era aceitável, pois iria demorar mais de oito horas para cada número de programadores. Com isto, com o primeiro algoritmo foi possível, em tempos “decentes”, alcançar um máximo de 36 tarefas. Acredito que seria possível chegar a, pelo menos, 37 tarefas ou até mesmo 38 caso tivesse definido um máximo de 8 programadores. Obtive os seguintes resultados para 36 tarefas:

T	P	Solution Time (em segundos)	Solution Time (em horas)
36	1	12360	3.43
36	2	13900	3.86
36	3	15000	4.17
36	4	16770	4.66
36	5	16570	4.60
36	6	18620	5.17
36	7	19470	5.40
36	8	22620	6.28
36	9	25450	7.06
36	10	29200	8.11

Obs: Este algoritmo foi rodado enquanto guardava várias informações em ficheiros e num segundo computador, um pouco antigo com um intel i3 350M no Linux Mint.

Ignorando o Lucro

Rodando agora o programa ignorando os lucros e apenas focando em fazer o número máximo de tarefas possível os resultados foram os seguintes (para 32 tarefas → linhas, com um máximo de 8 programadores → colunas):

MaxTasks	1	2	3	4	5	6	7	8
1	1	-	-	-	-	-	-	-
2	1	2	-	-	-	-	-	-
3	2	2	3	-	-	-	-	-
4	2	2	3	4	-	-	-	-
5	2	2	3	4	5	-	-	-
6	4	3	4	4	6	6	-	-
7	3	4	4	5	5	6	7	-
8	6	5	3	5	8	8	8	8
9	5	6	4	5	6	8	9	9
10	3	5	5	6	8	9	9	10
11	3	6	7	6	8	7	11	11
12	6	6	8	8	7	9	9	10
13	6	5	7	8	10	8	9	10
14	7	8	8	6	8	8	10	11
15	6	8	8	9	9	10	10	11
16	7	9	9	9	7	13	9	11
17	7	8	11	10	11	9	11	13
18	8	9	10	11	10	9	13	16
19	9	9	11	10	11	11	12	13
20	11	11	11	11	9	10	13	13
21	8	13	12	14	12	11	11	15

22	10	10	12	17	15	15	14	13
23	11	11	10	16	16	14	14	18
24	11	13	13	17	16	16	16	15
25	13	13	13	14	13	18	13	16
26	13	11	14	14	16	16	17	14
27	11	13	17	16	13	14	15	18
28	12	15	16	17	14	16	18	13
29	12	17	15	16	18	18	17	16
30	13	17	16	18	16	18	23	17
31	14	16	14	21	17	20	19	19
32	15	17	18	20	19	21	22	22

Para obter estes resultados foram necessárias fazer algumas alterações ao código anteriormente usado. Em primeiro lugar, comentei a secção do código onde calculo o lucro e verifico se é um novo melhor lucro, na função “solution” e acrescentei as seguintes linhas de código:

```
//Checking the max number of tasks that can be done
 int sum, loop;

 sum = 0;

 for(loop = problem->T - 1; loop >= 0; loop--) {
   sum = sum + jobs[loop];
 }
 if (sum > problem->maxTasks) problem->maxTasks = sum;
//end of checking the max number of tasks that can be done
```

Bem como, adicionar uma nova variável ao struct “problem”, denominada de maxTasks. Rodando o programa com “./job_selection_do_all.bash | tee output.txt” foi possível guardar as informações no ficheiro “output.txt” para depois analisar e formatar na tabela anterior.

Tasks válidas

Para calcular o número de tasks válidas bastou adicionar a variável *int viableCount* à struct *problem_t*. Incrementando esta variável a cada verificação de um set de tasks válido foi possível determinar o número de tasks válidas para cada conjunto de tarefas e programadores. Na secção "informações adicionais" deste relatório estão apresentados os resultados desta mudança, bem como de variáveis adicionais que foram acrescentadas à struct para obter diversas informações, a meu ver, relevantes para o problema do job selection.

Histograma de Lucros Possíveis

De forma a formar um histograma com todos os lucros obtidos usei as seguintes funções no programa em C:

```
#define HIST_SIZE 100000

long hist[HIST_SIZE];

void init_hist(void)
{
  for (int i = 0 ; i < HIST_SIZE; i++)
    hist[i] = 0l;
}

void add_to_hist(int profit)
{
  if (profit<0) profit = 0;
  if (profit >= HIST_SIZE) profit = HIST_SIZE - 1;
  hist[profit]++;
}

void write_hist(void)
{
  FILE *histData = fopen("histData.txt", "w");
  if(histData == NULL)
  {
    fprintf(stderr,"Unable to create file\n");
    exit(1);
  }
  for (int i = 0; i < HIST_SIZE; i++)
```

```
    if (hist[i] != 0l)
      fprintf(histData, "%d %ld\n",i,hist[i]);
}
```

O output necessário será escrito no ficheiro “histData.txt” e lido pelo Matlab da seguinte forma:

```
file = 'histData.txt';
data = load(file);
histogram(data(:,1), 'Normalization', 'count');
ylim([0 600]);
xlim([0 22000]);
legend('Number of Values in range');
legend show
```

Como o ficheiro “histData.txt” tem a seguinte estrutura:

```
7548 11
7549 11
7550 17
7551 15
7552 17
7553 11
7554 12
7555 10
7556 11
7557 4
```

Onde a primeira coluna representa o total profit e a segunda o número de vezes que este profit foi obtido.

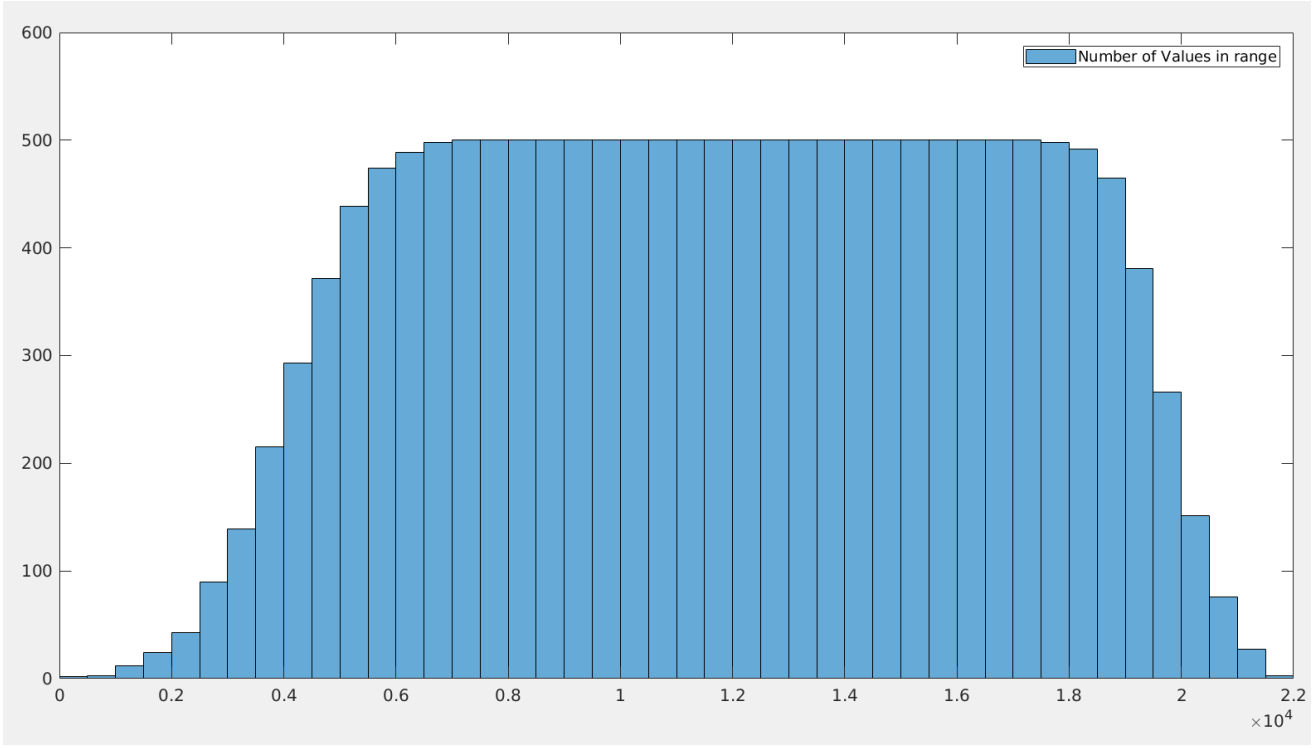
Foi possível facilmente calcular, de uma diferente forma, o número de sets de tasks válidos apenas efetuando a soma da segunda coluna de dados da seguinte forma:

```
Valid_combinations = sum(data(:,2))
```

O que no caso de 20 tarefas com 4 programadores deu:

```
Valid_combinations =

      238590
```

Para o histograma o resultado foi o seguinte:



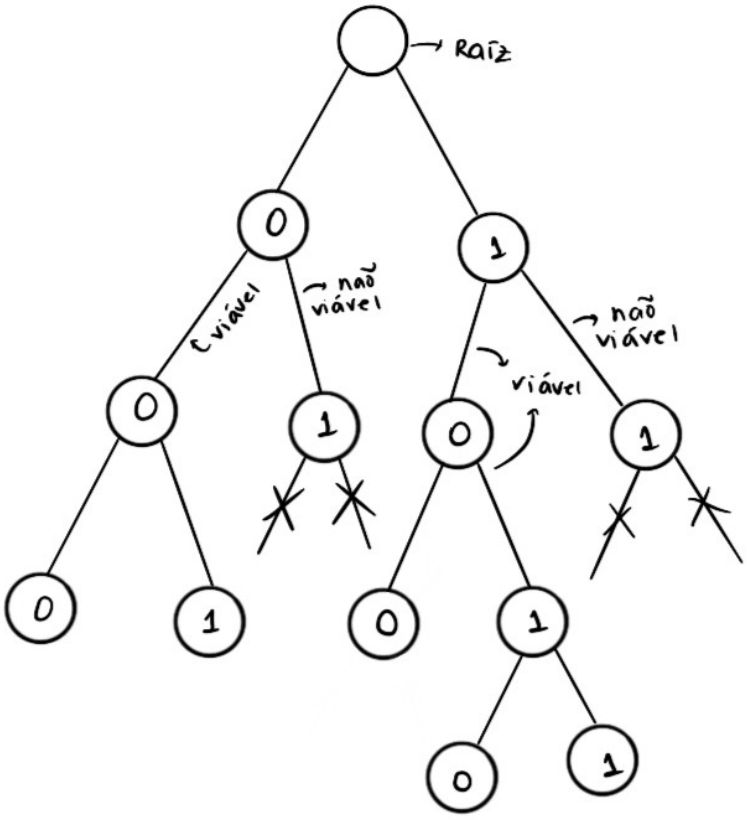
Podemos ver através da visualização do histograma que o mesmo apresenta uma distribuição, aproximadamente, normal.

Melhoramento do Código

Para melhorar o tempo de execução do algoritmo e não recorrer à "força bruta" para a sua resolução foi necessário repensar a maneira de resolver o problema.

Primeiro pensamento

O meu primeiro pensamento para um algoritmo mais eficaz em termos de tempo foi pensar nas possibilidades de set de tarefas como se se tratasse de uma árvore binária, na qual iria percorrer os nós e verificar se cada caminho é viável, recursivamente, caso seja, guardamos esse set de tarefas num array, caso contrário bastaria "cortar" este ramo da árvore e não avançar mais por aquele caminho, eliminando logo à partida muitas possibilidades de tarefas que já sabemos que não serão viáveis.



Segundo pensamento

O meu segundo pensamento, o qual acho que seja muito menos eficaz que o primeiro, seria de ordenar um array com todos os sets de tarefas por ordem decrescente de lucro. Percorrer o array até encontrar o primeiro set viável e terminar o programa pois foi encontrado o maior lucro possível. Mas. pensando bem, neste algoritmo o tempo seria imprevisível pois poderia ter sorte e ser os primeiros sets ou acabar por percorrer quase toda a lista. Já para não falar que teria de calcular o profit para todas as possibilidades.

Divide & Conquer

Esta técnica consiste em dividir um problema maior recursivamente em problemas menores até que o problema possa ser resolvido diretamente. Então a solução do problema inicial é dada através da combinação dos resultados de todos os problemas menores computados. Contudo, esta técnica não se adequa ao problema que nos deparamos num job selection. Para resolver o job selection dependemos sempre das restantes tasks para poder, ou não, efetuar outra task. A efetuação de cada uma das tasks é dependente uma das outras, logo não é possível resolver este problema com este método

Dynamic programming

Programação dinâmica é um método para a construção de algoritmos para a resolução de problemas computacionais, em especial os de otimização combinatória. É aplicável a problemas nos quais a solução ótima pode ser computada a partir da solução ótima previamente calculada e memorizada, de forma a evitar o recálculo de outros subproblemas que, sobrepostos, compõem o problema original.

O que um problema de otimização deve ter para que a programação dinâmica seja aplicável são duas principais características: subestrutura ótima e superposição de subproblemas.

Um problema apresenta uma subestrutura ótima quando uma solução ótima para o problema contém no seu interior soluções ótimas para subproblemas. A superposição de subproblemas acontece quando um algoritmo recursivo reexamina o mesmo problema muitas vezes.

O Job Selection Problem verifica as duas características para que a programação dinâmica seja aplicável, logo é possível resolver este programa usando este método.

Implementação do novo algoritmo

Com isto, decidi tentar solucionar o meu primeiro pensamento. Contudo, mudei algumas ideologias para tornarem o código ainda mais eficiente. Decidi que não faria sentido guardar os sets de tarefas viáveis num array e que poderia calcular de vez o profit e verificar se é maior que o best profit, caso seja viável e depois disto, sim, chamar novamente a função para o nível seguinte (caso sejam respeitadas as condições necessárias). Este método acabou por, furtuitamente, usar o método de programação dinâmica acima explicado, o que só mais tarde me apercebi.

Como esta abordagem envolve recorrer a árvores binárias tive de acrescentar ao programa as diferentes secções de código seguintes:

```
struct node
{
  int key_value;
  struct node *left;
  struct node *right;
};

struct node* newNode(int data, struct node* root)
{
    struct node* node = (struct node*)malloc(sizeof(struct node));
    node->key_value = data;
    node->left = node->right = NULL;
    return (node);
}
```

```
struct node* init_tree(struct node* root, int value, int control)
{
    struct node* temp = newNode(value, root);
    root = temp;
    if (control == 1)
    {
        // insert left child
      root->left = init_tree(root->left, 0, 0);


        // insert right child
      root->right = init_tree(root->right, 1, 0);
    }
    return root;
}
```

Na função “solve” é necessário alterar o seu conteúdo pois devemos alterar algumas variáveis durante a sua execução:

```
static void solve(problem_t *problem)
{
  FILE *fp;
  int i;

  //
  // open log file
  //
  (void)mkdir(problem->dir_name,S_IRUSR | S_IWUSR | S_IXUSR);
  fp = fopen(problem->file_name,"w");
  if(fp == NULL)
  {
    fprintf(stderr,"Unable to create file %s (maybe it already exists? If so,
delete it!)\n",problem->file_name);
    exit(1);
  }
  //
  // solve
  //
  problem->cpu_time = cpu_time();
    // call your (recursive?) function to solve the problem here
  int bestJobs[problem->T];
  struct node* tree = NULL;
  problem->viableCount = 0;
  problem->nCases = 0;
  for (int j = 0; j < problem->P; j++) problem->busy[j] = -1;
  for (int j = 0; j < problem->T-1; j++) problem->task[j].assigned_to = -1;
  tree = init_tree(tree, -1, 1);
  solution(tree->left, 0, bestJobs, problem);
  //Reset the important variables for another evaluation of combinations
  problem->total_profit = 0;
```

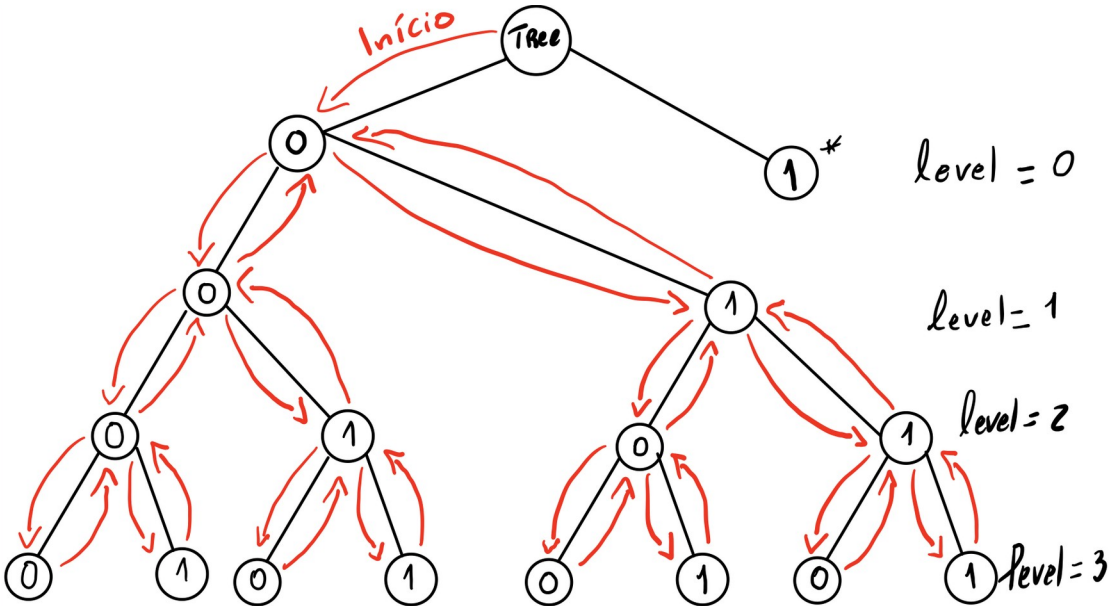
```
  for (int j = 0; j < problem->P; j++) problem->busy[j] = -1;
  for (int j = 0; j < problem->T-1; j++) problem->task[j].assigned_to = -1;
  //
  solution(tree->right, 0, bestJobs, problem);
  printf("Best Total Profit: %d", problem->bestTotalProfit);
  problem->cpu_time = cpu_time() - problem->cpu_time;
  //
  // save solution data
  //
  fprintf(fp,"NMec = %d\n",problem->NMec);
  fprintf(fp,"T = %d\n",problem->T);
  fprintf(fp,"P = %d\n",problem->P);
  fprintf(fp,"Profits%s ignored\n",(problem->I == 0) ? " not" : "");
  fprintf(fp,"Solution time = %.3e\n",problem->cpu_time);
  fprintf(fp,"Task data\n");
#define TASK  problem->task[i]
  for(i = 0;i < problem->T;i++)
    fprintf(fp,"  %3d %3d %5d\
n",TASK.starting_date,TASK.ending_date,TASK.profit);
#undef TASK
  fprintf(fp,"End\n");
.. }
```

Para o desenvolvimento da função “solution” recorri a um pensamento recursivo que percorre todos os nós da árvore começando na raiz da “Tree”. Após a chamada de tree→left e antes da chamada de tree→right reiniciamos todas as variáveis para avaliar o problema nas duas situações:

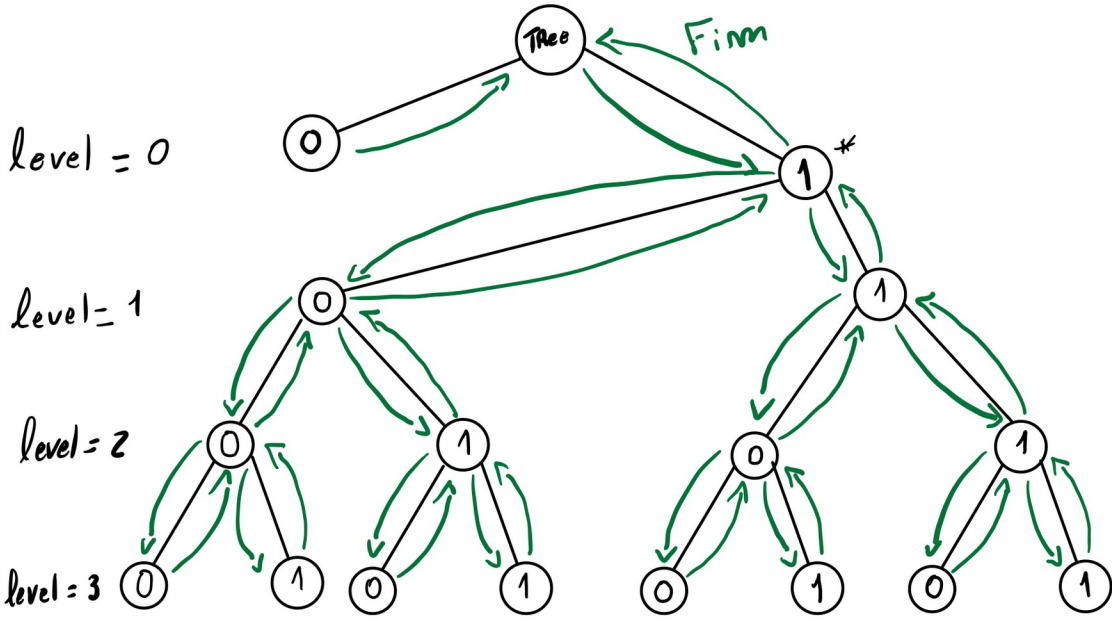
```
  solution(tree->left, 0, bestJobs, problem);

  //Reset the important variables for another evaluation of combinations
  problem->total_profit = 0;
  for (int j = 0; j < problem->P; j++) problem->busy[j] = -1;
  for (int j = 0; j < problem->T-1; j++) problem->task[j].assigned_to = -1;
  //

  solution(tree->right, 0, bestJobs, problem);
```

As figuras abaixo representam o percurso da função recursiva pelos nós da àrvore quando o número de tarefas é igual a 4:

- **solution(tree->left, 0, bestJobs, problem);**



- **solution(tree->right, 0, bestJobs, problem);**



A variável “problem” é partilhada por todas as chamadas da função e modificada em certos casos, devido a isto é necessário repor as suas variáveis quando voltamos ao “level” anterior da árvore para poder verificar as restantes possibilidades de tasks. Sempre que descemos um no “level” é necessário repor as váriaveis de problem, total_profit, busy e assigned_to aos valores do nó em que se encontram, pois estas variáveis estando dependentes das tasks e suas combinações são alterados ao longo das chamadas da função. A reposição destas variáveis será feita com recurso a variáveis temporárias em que cada função terá os seus valores nessas variáveis sendo que as mesmas são declaradas dentro da função e dependendo do momento e/ou valor do nó da árvore a atribuição de um valor às variáveis, bem como a sua reposição será diferente:

- **if (node->key_value == 1)**

Se esta condição se confirmar então teremos de verificar, em primeiro lugar, se é possível realizar esta tarefa, para isto desenvolvi o seguinte código:

```
if (node->key_value == 1)

  {
    for (int p = 0; p < problem->P; p++)
    {
      if (problem->busy[p] < problem->task[level].starting_date) //it’s viable
      {
        problem->task[level].assigned_to = p;
        problem->busy[p] = problem->task[level].ending_date;
        //profit calculation
        problem->total_profit += problem->task[level].profit;
        //end of profit calculation
        //check if new profit > than the actual best Profit
        if (problem->total_profit > problem->bestTotalProfit)
        {
          problem->bestTotalProfit = problem->total_profit;
          //change new best profit combination
          for (int k = 0; k < problem->T; k++)
          {
            if (problem->task[k].assigned_to >= 0)
              bestJobs[k] = 1;
            else
              bestJobs[k] = 0;
          }
        }
        viable = true;
        break;
```

```
    }

  else //the job is not viable

    viable = false;

    // end of assigned to phase
  }
```

- **Reposição das Variáveis**

Deveremos agora, antes de chamar recursivamente a função guardar os valores das variáveis no momento do nó em que nos encontramos:

```
  //Reset all important variables
  tempProfit = problem->total_profit;
  for (int i = 0; i < problem->T; i++) tempBusy[i] = problem->busy[i];
  for (int i = 0; i < problem->P; i++) tempAssignedTo[i] = problem-
>task[i].assigned_to;
  //
```

- **if (++level < problem->T && viable)**

Se esta condição se confirmar significa que teremos de continuar a formar sub-àrvores e chamar a função tanto para a “left child” como para a “right child”, caso contrário, as chamadas recursivas, do nó em questão são terminadas, voltando ao seu “parent”. Neste “if” temos dois momentos de reposição de variáveis:

- Depois da chamada para “node→left”:

```
    solution(node->left, level, bestJobs, problem);
    problem->total_profit = tempProfit;
    for (int i = 0; i < problem->T; i++) problem->busy[i] = tempBusy[i];
    for (int i = 0; i < problem->P; i++) problem->task[i].assigned_to =
tempAssignedTo[i];
```

É importante redefinir as váriaveis gerais depois da chamada para “node→left” e antes da chamada para “node→right” porque na chamada para a esquerda as variáveis podem ter sido alteradas e para analisar o caminho para o “node→right” temos de considerar o estado em que as variáveis se encontram no “parent node”.

- Após chamada para “node→right”:

```
solution(node->right, level, bestJobs, problem);
problem->total_profit = tempProfit;
for (int i = 0; i < problem->T; i++) problem->busy[i] = tempBusy[i];
for (int i = 0; i < problem->P; i++) problem->task[i].assigned_to =
tempAssignedTo[i];
```

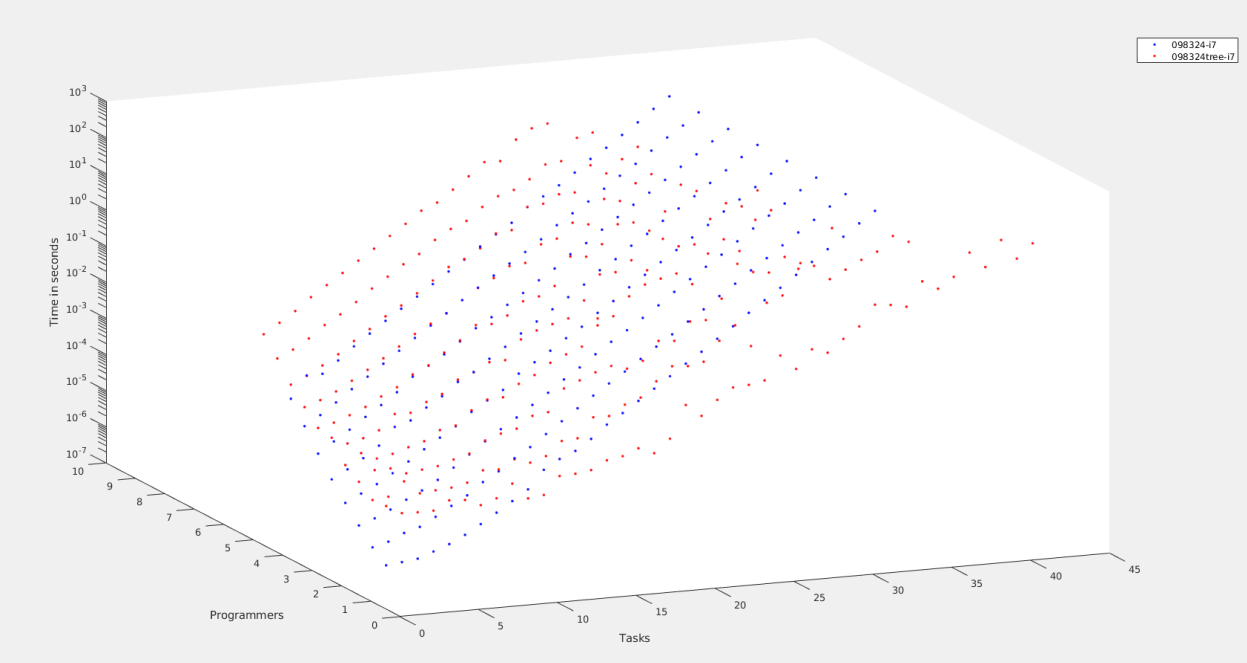
É necessário alterar as váriaveis depois da chamada para “node→right” porque nessa chamada as variáveis podem ter sido alteradas, logo temos de modificar os valores das variáveis que foram designadas na chamada efetuada para estarem resetadas quando voltarmos ao “level” abaixo do que nos encontramos, pois após esta redefinição a função no nó em questão terminará e iremos retornar ao nó “parent”.

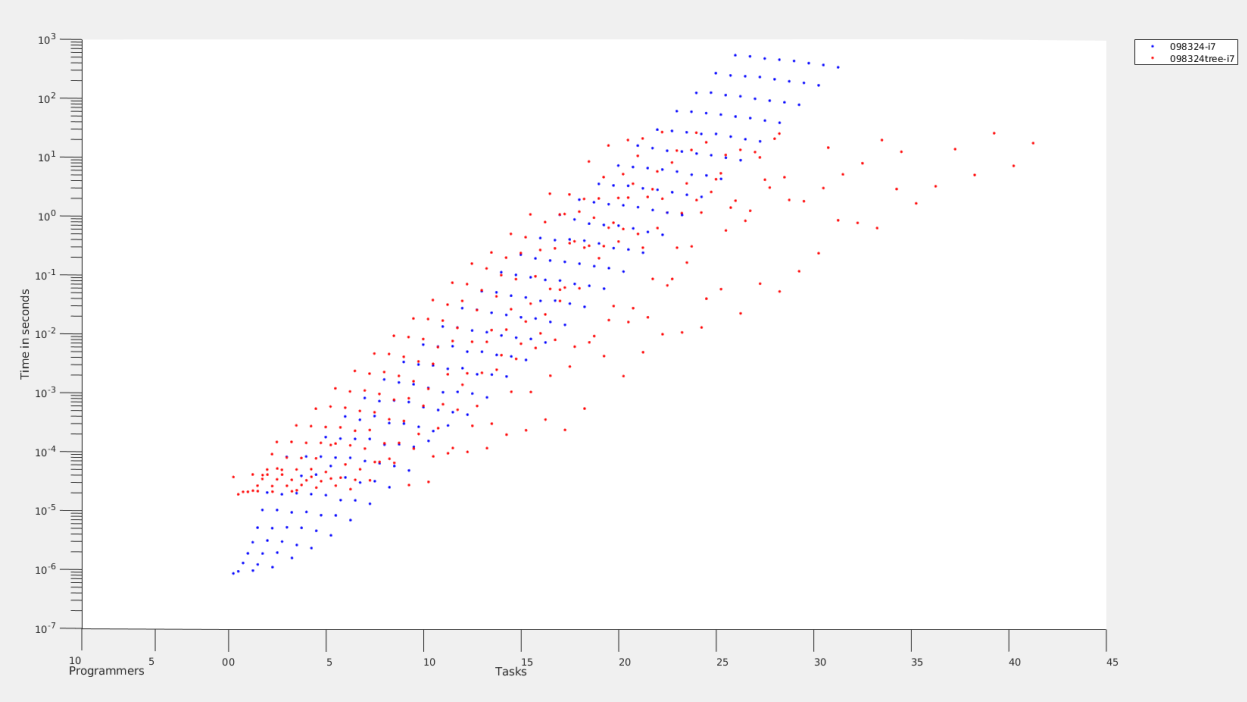
O objetivo desta função é que a mesma pare de ser chamada recursivamente quando o nó for uma “leaf” ou quando esse nó se revele não viável, não há razões para continuar por aquele caminho e formar sub-àrvores naquele nó caso o mesmo não forme uma combinação viável, poupando assim tempo perdido em combinações que já temos conhecimento, à partida, de que não serão viáveis.

Correndo este novo algoritmo agora num computador com um i7-9750H até 50 tarefas com 10 programadores e depois comparando com o algoritmo anterior, também corrido neste i7, mas desta vez, apenas até 32 tarefas com 8 programadores, obtive os seguintes resultados:

- Brute Force

- Método Novo





Podemos observar que com o novo método o tempo de execução acaba por ser superior em problemas de pequena dimensão, mas quando se trata de problemas de maiores dimensões o novo algoritmo supera, em muito, o anterior. Para ter ideia da diferença de tempo fiz cálculos para obter a percentagem de diminuição de tempo média.

Os cálculos efetuados verificaram que o tempo total de execução dos problemas reduziu em, aproximadamente, 185%. Verificamos também através da visualização do gráfico que este algoritmo não apresenta um tempo de execução linear. Para o mesmo número de tarefas o tempo aumenta com o aumento do número de programadores, correr o algoritmo para apenas 1 programador é extremamente rápido mesmo com uma grande quantidade de tarefas.

Como dito acima corri o programa num i7-9750H com o novo algoritmo até um máximo de 50 tarefas com 10 programadores. O algoritmo correu até ao fim, só que revelou alguns problemas, a partir de uma certa dimensão dependendo do número de programadores o programa foi “killed” pelo computador. O maior número de tarefas para o qual foi possível obter soluções foi para 42 tarefas com 1 programador, sendo que o programa foi parado para os restantes números de programadores, o mesmo aconteceu para 37, 38, 39, 40 e 41 tarefas, para números mais pequenos que estes, em alguns, foi apenas possível chegar até 2 programadores ou até mais. Pelo que percebi, efetuando uma pesquisa, o problema está na memória que reservei para o swap file do Linux no momento da sua instalação. Os resultados para as dimensões superiores a 32 tarefas que foram possíveis calcular foram os seguintes:

T	P	Best Profit	Tempo de execução (segundos)
33	1	34304	8.021e-01
33	2	47770	5.330e+00
33	3	50803	1.506e+01
34	1	49615	6.564e-01
34	2	45262	8.220e+00

35	1	59156	3.015e+00
35	2	53119	2.034e+01
36	1	52133	1.725e+00
36	2	42838	1.285e+01
37	1	51097	3.359e+00
38	1	79236	1.433e+01
39	1	39113	5.224e+00
40	1	49242	2.674e+01
41	1	61162	7.462e+00
42	1	64131	1.813e+01

Gerar combinações aleatórias

O objetivo desta “experiência” é gerar N combinações de tasks e verificar o profit desta mesmas combinação, comparando os valores obtidos com os do problema inicial. Será que é possível obtermos os mesmos resultados sem verificar todas as possibilidades?

Para solucionar este problema foi necessário gerar as possibilidades de forma diferente, para isto desenvolvi o seguinte código:

```
  for (int i = 0; i < 1000000; i++)
    solution(problem);

/* This function solves the main problem */

void solution(problem_t *problem)
{
  int randomNumber1, randomNumber2;
  for (int i = 0; i < problem->T; i++)
    problem->jobs[i] = 0;
  for (int i = 0; i < problem->T; i++)
  {
    srand(rand());
    randomNumber1 = rand();
    randomNumber2 = rand();
    if (randomNumber2 < randomNumber1)
      problem->jobs[i] = 1;
  }
    // visit all
    // assigned to phase
    for (int j = 0; j < problem->P; j++)
        problem->busy[j] = -1;
    for (int j = 0; j < problem->T; j++) problem->task[j].assigned_to = -1;
    int assignedCount = 0;
    for (int j = 0; j < problem->T; j++)
    {

        if (problem->jobs[j] == 0)
        {
          problem->task[j].assigned_to = -1;
        }
        else if (problem->jobs[j] == 1)
        {
          for (int p = 0; p < problem->P; p++)
          {
            if (problem->busy[p] < problem->task[j].starting_date)
            {
```

```
              problem->task[j].assigned_to = p;
              problem->busy[p] = problem->task[j].ending_date;
              break;
            }
          }
        }
      }
      int sum = 0;
      for (int j = 0; j < problem->T; j++)
      {
        sum += problem->jobs[j]; //sum will give us the number of jobs to do
        if (problem->task[j].assigned_to >= 0)
          assignedCount++; //assigned count will give us the number os tasks
with programmers
      }
      if (assignedCount == sum) // the job is viable
      {
      //profit calculation
        problem->total_profit = 0;
        for (int j = 0; j < problem->T; j++)
        {
          if (problem->task[j].assigned_to >= 0)
            problem->total_profit += problem->task[j].profit;
        }
      //end of profit calculation
      //check if new profit > than the actual best Profit
        if (problem->total_profit > problem->bestTotalProfit)
        {
          problem->bestTotalProfit = problem->total_profit;
        }
      //end of checking the best new profit
      }
    // end of assigned to phase
    return;
}
```

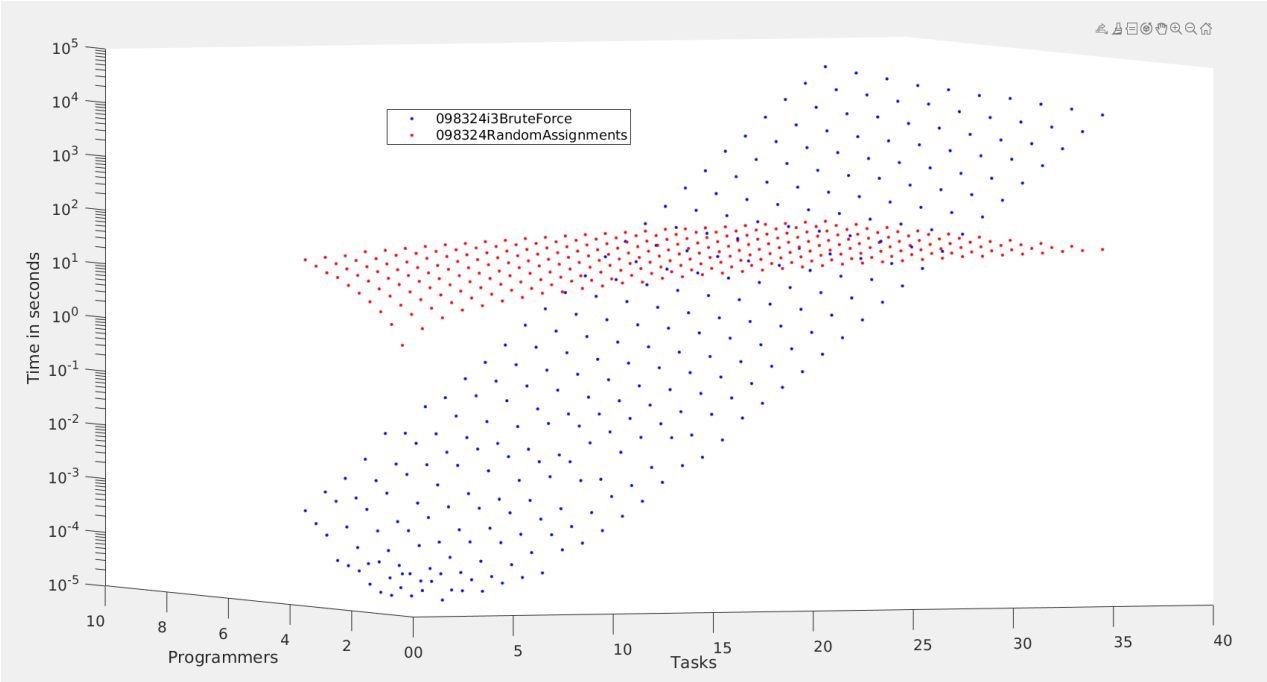
Para definir uma tarefa será ou não feita, gerei dois números aleatórios e caso o “randomNumber2” for menor que o “randomNumber1”, então a tarefa será realizada. Usei dois números aleatórios e não uma probabilidade fixa de fazer a tarefa para manter uma distribuição completamente aleatória. Analisando os resultados obtidos, verifiquei que, aproximadamente, apenas 33,6% dos melhores lucros estavam corretos, logo este método não é muito viável para muitas tarefas. Seriam necessárias gerar muito mais possibilidades para aumentar a percentagem de melhores lucros corretos. O tempo de execução apresenta um crescimento menor, mesmo em problemas com maiores números de tarefas sendo sempre, aproximadamente, igual ao problema anterior. Os

resultados completos deste programa estão apresentados numa tabela no Apêndice deste relatório.

O gráfico de comparação entre os tempos de Brute Force com os Random Assignments ficou da seguinte forma:

- Brute Force

- Random Assignment



Este resultado era previsível, visto que, independentemente, do número de tarefas escolhido serão sempre geradas um milhão de possibilidades. Daí mesmo o caso de apenas uma tarefa não estar muito distante do caso com 36 tarefas em termos de tempo de execução.

Informações adicionais

Para obter informações adicionais acrescentei as seguintes linhas de código na função “solve” dada pelo professor:

```
  //
  // solve
  //
  problem->cpu_time = cpu_time();
 int jobs[problem->T], bestJobs[problem->T];
 problem->viableCount = 0;
 problem->nCases = 0;
 FILE *f = fopen("table098324.txt", "a");
 solution(jobs, 0, bestJobs, problem);
 fprintf(f, "%d\t%d\t%d\t%d\t%d\t", problem->T, problem->P, problem->nCases,
 problem->viableCount, problem->bestTotalProfit);
 for (int j = 0; j < problem->T; j++)
   fprintf(f, "%d", bestJobs[j]);
 fprintf(f, "\n");
 fclose(f);
  problem->cpu_time = cpu_time() - problem->cpu_time;
  //
```

Estas linhas de código guardam no ficheiro “table098324.txt” várias informações adicionais. Para além do número de tarefas (problem→T) e do número de programadores (problem→P) para identificar o problema que estamos a tratar, também guarda o número de casos totais para as várias tarefas, que acaba por ser igual a 2^problem->T, sejam viáveis ou não (problem→nCases) incrementando a variável em cada novo set de tarefas, o número de casos viáveis (problem→viableCount), incrementando a variável apenas quando o set de tarefas se mostra viável, o melhor lucro possível (problem→ bestTotalProfit), bem como o conjunto de tarefas em que obtemos esse melhor lucro (a partir do ciclo for e do array bestJobs).

No ficheiro “table098324.txt” ficaram guardadas as informações até 36 tarefas com um máximo de 10 programadores corridos no programa usando o método da “força bruta”:

T	P	nCases	viable	Best Profit	Best Set
1	1	2	2	4018	1
2	1	4	3	4958	10
2	2	4	4	3875	11
3	1	8	6	2875	110
...	...	...	...	...	...
36	10	0	533362018	47776	1111110010010011000001011000111100111

O “nCases” de 32 tarefas, ou mais, fica definido a 0 pois o número ultrapassa a gama de valores abrangida por um *int* (>2^31 - 1). Poderia ter evitado isto tendo definido a variável “nCases” como um *long* invés de *int*. Foi um pormenor que me passou ao lado aquando da realização do código e que para repetir o processo com *“long”* seriam necessárias muitas mais horas a correr o programa, o que achei desnecessário pois o valor correto de “nCases” pode ser obtido por 2^problem→T.

Todos os resultados obtidos neste ficheiro poderão ser consultados no Apêndice deste relatório.

“Surpresa”

A realizar este trabalho surgiu-me a curiosidade de qual seria o número mínimo de programadores (P) necessários para efetuar T tarefas. Para solucionar esta questão bastavam umas simples alterações no algoritmo. Por isso, decidi dedicar um pouco do tempo deste trabalho a solucionar esta ideia. Para tal, efetuei as seguintes alterações no ficheiro “job_selection_how_many_p.c”:

- Retirei a verificação do input de número de programadores;

- Alterei o limite das tarefas e de programadores para 123, pois foi o número máximo de tarefas que o programa deixou correr;
- Reduzi a função “solve” dada pelo professor eliminando todas as escritas de ficheiros e informações que não se relevaram importantes para o objetivo do novo programa, ficando com a seguinte função:

```
static void solve(problem_t *problem)
{
  int i;
  //
  // solve
  //
  problem->cpu_time = cpu_time();
    // call your (recursive?) function to solve the problem here
  int jobs[problem->T];
  for (i = 0; i<problem->T; i++) jobs[i] = 1;
  if (isViable(jobs, problem))
  {
    printf("We would need at least %d programmers to get all %d tasks done\n",
problem->P, problem->T);
    exit(0);
  }

  problem->cpu_time = cpu_time() - problem->cpu_time;
}
```

- Alterei também a função “main”, retirei a atribuição da variável P e defini “I” sempre a 0, pois o seu valor é irrelevante:

```
int main(int argc,char **argv)
{
  problem_t problem;
  int NMec,T,P,I;

  NMec = (argc < 2) ? 2020 : atoi(argv[1]);
  T = (argc < 3) ? 5 : atoi(argv[2]);
  I = 0;
  for(P = 1; P <= T; P++)
  {
    init_problem(NMec,T,P,I,&problem);
    solve(&problem);
  }
  return 0;
}
```

- Criei a função isViable que retorna um bool e é chamada na função “solve” da seguinte forma:

```
  if (isViable(jobs, problem))
  {
    printf("We would need at least %d programmers to get all %d tasks done\n",
problem->P, problem->T);
    exit(0);
  }

bool isViable(int jobs[], problem_t *problem)
{
    for (int j = 0; j < problem->P; j++) //Define all busy's to -1
      {
        problem->busy[j] = -1;
      }
    int assignedCount = 0; //Number of Tasks that were assigned to someone
    for (int j = 0; j < problem->T; j++)
    {
      for (int p = 0; p < problem->P; p++)
      {
        if (problem->busy[p] < problem->task[j].starting_date)
        {
          problem->task[j].assigned_to = p;
          problem->busy[p] = problem->task[j].ending_date;
          assignedCount += 1;
          break;
        }
      }
    }
    if (assignedCount == problem->T) // the job is viable
      return true;
    return false;
}
```

- Usando os conteúdos do script “job_selection_do_all.bash” e efetuando algumas alterações, criei um novo script “howmanyP_do_all.sh”:

```
#! /bin/bash

NMec=98324
I=1
d=$(printf "%06d" $NMec)
if [ -d $d ]; then
  d=$(printf "%06d" $NMec)
  rm -vf $(grep -L End $d/*)
```

```
fi
for T in {1..123}; do
    ./howManyP $NMec $T
done
```

- Por fim, rodei no terminal "./howManyP_do_all.sh | tee howManyP.txt" e guardei as informações obtidas no ficheiro "howManyP.txt" que estará anexado a este relatório, com o seguinte conteúdo até 123 tarefas:

```
We would need at least 1 programmers to get all 1 tasks done
We would need at least 2 programmers to get all 2 tasks done
We would need at least 3 programmers to get all 3 tasks done
We would need at least 4 programmers to get all 4 tasks done
We would need at least 5 programmers to get all 5 tasks done
We would need at least 5 programmers to get all 6 tasks done
We would need at least 7 programmers to get all 7 tasks done
We would need at least 5 programmers to get all 8 tasks done
We would need at least 7 programmers to get all 9 tasks done
We would need at least 8 programmers to get all 10 tasks done
We would need at least 7 programmers to get all 11 tasks done
We would need at least 9 programmers to get all 12 tasks done
We would need at least 9 programmers to get all 13 tasks done
We would need at least 11 programmers to get all 14 tasks done
We would need at least 10 programmers to get all 15 tasks done
We would need at least 13 programmers to get all 16 tasks done
We would need at least 12 programmers to get all 17 tasks done
We would need at least 13 programmers to get all 18 tasks done
We would need at least 14 programmers to get all 19 tasks done
We would need at least 16 programmers to get all 20 tasks done
We would need at least 14 programmers to get all 21 tasks done
We would need at least 17 programmers to get all 22 tasks done
We would need at least 17 programmers to get all 23 tasks done
We would need at least 17 programmers to get all 24 tasks done
We would need at least 15 programmers to get all 25 tasks done
We would need at least 19 programmers to get all 26 tasks done
```

Como as tarefas são criadas com base no número mecanográfico e no número de programadores as T tarefas em cada caso verificado serão diferentes. Pensei em alterar a forma como as tarefas são geradas para não incluírem o número de programadores, mas decidi deixar como estava e assim o programa devolve o primeiro caso de T tarefas para P programadores em que for possível realizar todas as tarefas.

Conclusão

Neste trabalho resolvemos um problema de seleção de trabalhos, no qual tivemos de definir diferentes tarefas e reparti-las por vários, ou apenas um programador, onde cada programador só podia realizar uma tarefa de cada vez. Na realização deste trabalho percebi o quão importante é ter cuidado com o processamento de um algoritmo e com a memória. Muitos dos meus pensamentos iniciais revelaram não ser possíveis devido a problemas de memória, daí ter de adaptar e tornar o algoritmo muito mais eficiente. A aplicação da programação dinâmica revelou-se extremamente eficiente ao algoritmo em questão. Certamente, a “força bruta” não será solução para problemas de otimização com esta dimensão.

Bibliografia

Wikipedia, «Programação Dinâmica», «Divide & Conquer», «Activity Selection problem»

zrzahid.com, «Weighted job/interval scheduling – Activity Selection Problem»

tutorialspoint, «Weighted Job Scheduling»

Estes *sites* foram usados para confirmar informação e/ou complementar o relatório.

Apêndice

Todos os ficheiros utilizados para a realização deste relatório podem ser consultados no Repositório do Git Hub (github.com/sousaUA/JobSelection).

Código do novo algoritmo

```
void solution(struct node *node, int level, int bestJobs[], problem_t
*problem)

{
  bool viable = true;
  int tempProfit = 0, tempAssignedTo[problem->T], tempBusy[problem->P];
  // visit all
  // assigned to phase
  if (node->key_value == 1)
  {
    for (int p = 0; p < problem->P; p++)
    {
      if (problem->busy[p] < problem->task[level].starting_date) //is viable
      {
        problem->task[level].assigned_to = p;
        problem->busy[p] = problem->task[level].ending_date;


        //profit calculation
        problem->total_profit += problem->task[level].profit;
        //end of profit calculation
        //check if new profit > than the actual best Profit
        if (problem->total_profit > problem->bestTotalProfit)
        {
          problem->bestTotalProfit = problem->total_profit;
          //change new best profit combination
          for (int k = 0; k < problem->T; k++)
          {
            if (problem->task[k].assigned_to >= 0)
              bestJobs[k] = 1;
            else
              bestJobs[k] = 0;
          }
        }
        viable = true;
        break;
      }
      else //the job is not viable
        viable = false;
      // end of assigned to phase
    }
  }
```

```
  else if (node->key_value != 0 && node->key_value != 1)
    printf("Strange... This shouldn't appear in your screen... '");
  //Reset all important variables
  tempProfit = problem->total_profit;
  for (int i = 0; i < problem->T; i++) tempBusy[i] = problem->busy[i];
  for (int i = 0; i < problem->P; i++) tempAssignedTo[i] = problem-
>task[i].assigned_to;
  //
  if (++level < problem->T && viable)
  {
    node = init_tree(node, node->key_value, 1);
    solution(node->left, level, bestJobs, problem);
    //Reset all important variables
    problem->total_profit = tempProfit;
    for (int i = 0; i < problem->T; i++) problem->busy[i] = tempBusy[i];
    for (int i = 0; i < problem->P; i++) problem->task[i].assigned_to =
tempAssignedTo[i];
    //
    solution(node->right, level, bestJobs, problem);
    //Reset all important variables
    problem->total_profit = tempProfit;
    for (int i = 0; i < problem->T; i++) problem->busy[i] = tempBusy[i];
    for (int i = 0; i < problem->P; i++) problem->task[i].assigned_to =
tempAssignedTo[i];
  }
  return;
}
```

Código de “job_selection_how_many_P.c”

```
bool isViable(int jobs[], problem_t *problem)

{
    for (int j = 0; j < problem->P; j++) //Define all busy's to -1
      {
        problem->busy[j] = -1;
      }
    int assignedCount = 0; //Number of Tasks that were assigned to someone
    for (int j = 0; j < problem->T; j++)
    {
      for (int p = 0; p < problem->P; p++)
      {
        if (problem->busy[p] < problem->task[j].starting_date)
        {
          problem->task[j].assigned_to = p;
          problem->busy[p] = problem->task[j].ending_date;
          assignedCount += 1;
```

```
          break;
        }
      }
    }
    if (assignedCount == problem->T) // the job is viable
      return true;
    return false;
}
////////////////////////////////////////////////////////////////////////


#if 1


static void solve(problem_t *problem)
{
  int i;
  //
  // solve
  //
  problem->cpu_time = cpu_time();
    // call your (recursive?) function to solve the problem here
  int jobs[problem->T];
  for (i = 0; i<problem->T; i++) jobs[i] = 1;
  if (isViable(jobs, problem))
  {
    printf("We would need at least %d programmers to get all %d tasks done\n",
problem->P, problem->T);
    exit(0);
  }

  problem->cpu_time = cpu_time() - problem->cpu_time;
}
#endif
////////////////////////////////////////////////////////////////////////
// main program
int main(int argc,char **argv)
{
  problem_t problem;
  int NMec,T,P,I;


  NMec = (argc < 2) ? 2020 : atoi(argv[1]);
  T = (argc < 3) ? 5 : atoi(argv[2]);
  I = 0;
  for(P = 1; P <= T; P++)
  {
    init_problem(NMec,T,P,I,&problem);
    solve(&problem);
  }
  return 0;
```

```
}
```

Resultados de “howManyP_do_all.sh”

Este programa retorna o número mínimo de programadores necessários para realizar todas as T tarefas. Estes foram os resultados de {1..123} tarefas:

Número de Tarefas	Programadores necessários para realizar todas as tarefas	Número de Tarefas	Programadores necessários para realizar todas as tarefas
1	1	63	45
2	2	64	44
3	3	65	44
4	4	66	44
5	5	67	40
6	5	68	48
7	7	69	46
8	5	70	48
9	7	71	47
10	8	72	45
11	7	73	49
12	9	74	48
13	9	75	51
14	11	76	52
15	10	77	50
16	13	78	54
17	12	79	54
18	13	80	55

19	14	81	58
20	16	82	57
21	14	83	51
22	17	84	53
23	17	85	60
24	17	86	59
25	15	87	56
26	19	88	58
27	20	89	59
28	22	90	56
29	23	91	60
30	22	92	61
31	19	93	64
32	21	94	60
33	25	95	62
34	25	96	65
35	22	97	64
36	23	98	65
37	27	99	59
38	26	100	67
39	27	101	64
40	28	102	68
41	27	103	63
42	26	104	68

43	29	105	65
44	29	106	73
45	30	107	66
46	29	108	74
47	31	109	72
48	34	110	72
49	33	111	73
50	33	112	74
51	35	113	67
52	31	114	78
53	36	115	73
54	37	116	77
55	35	117	82
56	35	118	79
57	38	119	79
58	40	120	80
59	41	121	81
60	43	122	81
61	42	123	77
62	41		

Resultados de “table098324.txt”

Tasks	Programmers	nCases	viable	Best Profit	Best Set of Tasks
1	1	2	2	4018	1
2	1	4	3	4958	10
2	2	4	4	3875	11
3	1	8	6	2875	110
3	2	8	7	2634	110
3	3	8	8	4517	111
4	1	16	7	4771	1000
4	2	16	11	6071	1100
4	3	16	15	7058	1110
4	4	16	16	10226	1111
5	1	32	12	3626	01010
5	2	32	16	7630	11000
5	3	32	26	7088	11100
5	4	32	31	9983	11110
5	5	32	32	10528	11111
6	1	64	28	4100	101001
6	2	64	26	8443	100100
6	3	64	52	12035	110110
6	4	64	57	10331	111010
6	5	64	64	11795	111111
6	6	64	64	9791	111111
7	1	128	22	9889	1001000

7	2	128	55	6004	0110010
7	3	128	84	11578	1110100
7	4	128	114	9516	1110101
7	5	128	120	9170	1101110
7	6	128	127	14163	1111101
7	7	128	128	16936	1111111
8	1	256	90	7753	01001101
8	2	256	121	8355	11000110
8	3	256	93	7495	11000001
8	4	256	197	12636	11110001
8	5	256	256	12773	11111111
8	6	256	256	19065	11111111
8	7	256	256	16852	11111111
8	8	256	256	14370	11111111
9	1	512	110	11276	110001010
9	2	512	202	9263	110011110
9	3	512	151	8820	011010001
9	4	512	321	13348	110100010
9	5	512	438	20895	111000111
9	6	512	508	12954	111111101
9	7	512	512	13862	111111111
9	8	512	512	16473	111111111
9	9	512	512	21947	111111111
10	1	1024	38	18773	0010000000
10	2	1024	319	12222	1001011001
10	3	1024	330	14472	1100000101

10	4	1024	652	14267	1111101000
10	5	1024	960	12811	1111100111
10	6	1024	1016	17058	1110111111
10	7	1024	1020	19911	1111111011
10	8	1024	1024	17707	1111111111
10	9	1024	1024	21958	1111111111
10	10	1024	1024	28276	1111111111
11	1	2048	48	19939	10000000001
11	2	2048	560	13171	11001010001
11	3	2048	830	14041	10010010011
11	4	2048	779	15556	11011000001
11	5	2048	1696	18559	11111010100
11	6	2048	1695	19367	11001110101
11	7	2048	2048	11720	11111111111
11	8	2048	2048	17182	11111111111
11	9	2048	2048	19337	11111111111
11	10	2048	2048	25165	11111111111
12	1	4096	390	13740	010100110010
12	2	4096	322	14195	110000010001
12	3	4096	1411	13193	001010111001
12	4	4096	2544	15139	111110010010
12	5	4096	2508	18312	111011100001
12	6	4096	3666	21576	011111101101
12	7	4096	3736	21252	111011101011
12	8	4096	4052	22657	111111111100
12	9	4096	4096	25200	111111111111

12	10	4096	4094	21949	111111110111
13	1	8192	453	16129	1001001010000
13	2	8192	405	9924	1100001000100
13	3	8192	1113	20236	1101000000001
13	4	8192	3603	19475	1011010101010
13	5	8192	6097	17333	0011111010111
13	6	8192	5706	17998	1111100100011
13	7	8192	7024	20500	1110110111001
13	8	8192	7834	20819	1110011101111
13	9	8192	8192	26005	1111111111111
13	10	8192	8192	26165	1111111111111
14	1	16384	600	13187	11000000010110
14	2	16384	1912	15720	11010100101011
14	3	16384	3302	21990	11010100010011
14	4	16384	2515	16296	11011000010000
14	5	16384	7046	17365	11100110011000
14	6	16384	9872	26589	10101110100001
14	7	16384	13878	24725	11111010101001
14	8	16384	15728	27294	11111001011101
14	9	16384	16068	29226	01111011111011
14	10	16384	16366	30266	11111110111111
15	1	32768	714	19616	101000000100001
15	2	32768	1736	16906	110001001010001
15	3	32768	4288	18780	100101000010111
15	4	32768	8457	20842	001011010000010
15	5	32768	13572	15621	111001001100101

15	6	32768	23058	20716	101101011010110
15	7	32768	24688	25720	111110100001111
15	8	32768	29786	28764	111111110001101
15	9	32768	32096	23895	111101111011011
15	10	32768	32768	28229	111111111111111
16	1	65536	876	18246	1001000101000001
16	2	65536	5966	17888	1000110101011010
16	3	65536	14328	18231	1110100100101010
16	4	65536	11539	16714	1110100010010001
16	5	65536	11860	31576	1111100000000001
16	6	65536	56718	19371	1111011001111111
16	7	65536	39392	24270	1111111000000011
16	8	65536	56746	31656	1111110001101101
16	9	65536	59540	37845	1111100100011111
16	10	65536	65308	25888	1111111111101011
17	1	131072	1640	29416	01011000000000001
17	2	131072	6117	25110	00101000001001100
17	3	131072	27424	18832	10110101000001111
17	4	131072	29790	24459	11110101000001101
17	5	131072	45737	27604	10100111100001011
17	6	131072	41854	22487	01100110010010110
17	7	131072	90872	30485	10110111001100111
17	8	131072	118184	31082	11011011101011011
17	9	131072	126968	27496	01111111100101111
17	10	131072	130312	25868	01111111111110111
18	1	262144	1130	38215	010001000000000000

18	2	262144	9118	24939	001100001001011000
18	3	262144	33162	25773	101100101010100001
18	4	262144	45292	26389	101100011001000101
18	5	262144	70266	23971	111110001001100011
18	6	262144	61154	23383	101111010000000011
18	7	262144	169858	26539	110111101001000011
18	8	262144	258204	27564	111111011111011110
18	9	262144	217312	30138	111110010101100111
18	10	262144	261280	30078	111111110111101111
19	1	524288	3057	30117	0010000000000000101
19	2	524288	15192	24167	1110010000010000001
19	3	524288	44023	27660	0101010010010000110
19	4	524288	75056	26392	1110010000010111011
19	5	524288	106313	28602	1011110000000010101
19	6	524288	171916	25363	1110110100000100111
19	7	524288	302288	26657	1111110100011000111
19	8	524288	385772	28400	0110111110100001111
19	9	524288	477182	33769	1011101101101110111
19	10	524288	491780	30568	0110111111110001111
20	1	1048576	18560	23617	00100111011101010001
20	2	1048576	48275	36597	00110100100000000011
20	3	1048576	38181	27672	00110101000100100111
20	4	1048576	238590	21561	10100101010110010011
20	5	1048576	99828	23809	10101110010001010001
20	6	1048576	184883	27458	10110111000001000111
20	7	1048576	566240	31608	11101110100001101111

20	8	1048576	600357	25943	11111100110010001001
20	9	1048576	775728	25163	11011111010100101010
20	10	1048576	1031765	37617	11111011011101111110
21	1	2097152	8080	23870	100100101010000100100
21	2	2097152	118656	31382	101000100100000010011
21	3	2097152	63461	32060	110100000001000011011
21	4	2097152	425280	26823	111000101100011011101
21	5	2097152	415840	28038	110111001001100001111
21	6	2097152	381544	25030	110111100000001001111
21	7	2097152	608064	26445	010111101100001000111
21	8	2097152	1584060	31694	111101011011000001111
21	9	2097152	1197096	30359	010011101010110101011
21	10	2097152	1533456	34948	111111101000000101101
22	1	4194304	20162	24130	101110100010000011001
22	2	4194304	84402	31358	101000011110010010011
22	3	4194304	191492	30352	110100000110000000111
22	4	4194304	1294980	36391	111010111001010101001
22	5	4194304	1931169	25611	101001110011101010101
22	6	4194304	2133677	25028	001100111100011000001
22	7	4194304	1954432	24728	011100101001101101111
22	8	4194304	1728688	31584	010110000101111001011
22	9	4194304	2811200	26602	111101001011000111111
22	10	4194304	3132417	39154	110010001110101111101
23	1	8388608	49374	20814	010110001100001011100 11
23	2	8388608	111292	39677	110001000000000000010011
23	3	8388608	140088	43207	101011000001000000100001

23	4	8388608	2423820	38674	01111000110010011000101
23	5	8388608	2107476	27877	11100110011000110001111
23	6	8388608	2062944	34285	11101010010100011101011
23	7	8388608	2037887	31934	11101100110000101010101
23	8	8388608	7049866	31698	11011111100100111011011
23	9	8388608	4780616	35435	01010110111100011010111
23	10	8388608	6851624	31679	11111011100010001011111
24	1	16777216	55450	40687	10111000000000000000000
24	2	16777216	357272	30337	11001101000011000000110
24	3	16777216	516810	36206	01101000001100100010101 0
24	4	16777216	3129676	38088	10011110001110000110010 0
24	5	16777216	4127300	26699	10110100011100101011111 1
24	6	16777216	4162349	38406	11010110100010101100011 1
24	7	16777216	5996491	37833	10110100101010011100100 1
24	8	16777216	7243071	27055	11110101001011001000001 1
24	9	16777216	6478125	32816	11111001011001000010001 1
24	10	16777216	14984540	29560	11110111111110101100110 1
25	1	33554432	81418	52643	10100010000000000000000 0
25	2	33554432	893724	30361	11000000101110011001101 10
25	3	33554432	497424	26762	01010100000001000000001 01
25	4	33554432	3547054	37499	11010000101010011010100 01
25	5	33554432	1636348	35819	11111011100000010000001 11
25	6	33554432	12566985	32398	10011011010101011001101
25	7	33554432	4650696	36293	10110111010000000100001 11
25	8	33554432	12280776	39602	11111010001100010011011 01
25	9	33554432	12015734	34310	11011111101000000000101 101

25	10	33554432	13544275	38090	111100011100110000010011 1
26	1	67108864	244664	30742	100110000000000000000000101
26	2	67108864	153668	43764	110000000000000000000001000
26	3	67108864	1865749	40906	110100011000001000001110 10
26	4	67108864	1716100	42004	011010001000000001001000 11
26	5	67108864	11110632	39712	111001001010110110000101 01
26	6	67108864	12524413	39317	011101100100000000101011 10
26	7	67108864	22740448	34944	111010010101101000001111 11
26	8	67108864	12880686	39360	110110001011100000001011 11
26	9	67108864	28443253	37918	011111011100000100110111 11
26	10	67108864	52534296	36857	011111011000011101110111 11
27	1	134217728	81774	37757	010000000000101010000000000
27	2	134217728	2585408	31216	111110100100100001100110001
27	3	134217728	17241328	38810	111010101100000100001100011
27	4	134217728	11868596	40590	111101010000101101001001011
27	5	134217728	6566588	37187	101100110010000001101110001
27	6	134217728	7188532	37633	011101001000010010001011110
27	7	134217728	15388020	30287	011011010101100010001100101
27	8	134217728	67410754	38734	111010011110000111100010111
27	9	134217728	26262630	36579	011011000101110000111001001
27	10	134217728	98244528	39166	111110111000111011001001111
28	1	268435456	247872	32437	101011000011000100100000000 0
28	2	268435456	5526192	36272	110101100110000100000100011 1
28	3	268435456	6807092	45002	000111000010000100000100000 0
28	4	268435456	21987780	37461	100110101000110010010100001 0
28	5	268435456	7442536	38691	110000011010000100001101111 1

28	6	268435456	20565956	39418	11110000110100010001001000011
28	7	268435456	50274008	36970	11011110011000010001110011111
28	8	268435456	26983994	32532	11111101100000000010110000101
28	9	268435456	118495824	41687	11111100101100010010001011111
28	10	268435456	119739016	41476	10110010111001100001110111111
29	1	536870912	189810	47245	10000100000000000000000000001
29	2	536870912	15028304	36831	11110100111010010100110000011
29	3	536870912	6764256	44922	11100000001110001000001000111
29	4	536870912	11504960	34613	11011101000001100000011000111
29	5	536870912	34840092	51126	11111000000100010010011001101
29	6	536870912	110725702	46038	11110010101000110101000010011
29	7	536870912	82391233	46140	11111001000010001010010011001
29	8	536870912	78421766	36454	11110100001000110011000011011
29	9	536870912	144971270	37932	01101010111100100000000111101
29	10	536870912	335370444	42941	01110111111010001101000111111
30	1	1073741824	304596	36741	10100100100000000000000000001
30	2	1073741824	27565516	41011	11100011001000100110100100011
30	3	1073741824	17426735	42384	01110010100000100110010011001
30	4	1073741824	54168838	47985	01001100001011010010000001100
30	5	1073741824	66276706	51329	10011101010010101100100001000
30	6	1073741824	65182280	47955	11111100010000001000001110001
30	7	1073741824	534293272	38097	10111110100100011000110111111
30	8	1073741824	161235612	45837	10101011100100100010001000011
30	9	1073741824	463202010	41398	10010111101101110010100010111
30	10	1073741824	499257661	42820	11111101011000100011011100101
31	1	-2147483648	876582	33793	10001000000000000000110100011000

31	2	-2147483648	11367004	39237	10010000000110000100000001000101
31	3	-2147483648	12165664	52470	01100000010000000001000010000011
31	4	-2147483648	118000132	47737	11110010001011010011001001000110
31	5	-2147483648	52859954	43335	11100101010100101010011000000111
31	6	-2147483648	280909214	49555	11110100100000100011011000010111
31	7	-2147483648	274549444	44164	10111111100000101010000001110011
31	8	-2147483648	309763263	40808	01101101011001001010000010100011
31	9	-2147483648	203432772	38837	01001101111010011000010000000111
31	10	-2147483648	306846866	39029	10000110110010100001101011001000
32	1	0	4144822	42307	01000000000000000000000000100101
32	2	0	17627610	44666	11100001000001001001000000011101
32	3	0	168553788	41847	10110010111011001000110001010001
32	4	0	301739232	50484	11110100001101000011011001001011
32	5	0	160888612	37878	01011110000110001001010010001111
32	6	0	884925124	46567	11110110001010000110101101101111
32	7	0	1124344246	39889	11100011111001101100000110101011
32	8	0	1257090324	42146	11100111101100110000111010011111
32	9	0	1473057424	43814	10111101101001011101010000111101
32	10	0	-1954634965	46911	11011110001110010100101100111100
33	1	0	3919367	34304	00010000100000000000000000001011
33	2	0	20152569	47770	11000000000000000000000000000000
33	3	0	86642621	50803	11011000010110011000001010000100 1
33	4	0	192507118	52204	11101010100001010000000110000011
33	5	0	621218312	55356	01101101000000001000000000100010 1
33	6	0	1368836298	48753	11111011001001100100000010100111 1
33	7	0	-1529127972	54161	11001101110100000010010010111111 1

33	8	0	-102768400	46456	11111111110001011011010101111010 1
33	9	0	1117482458	46058	11111100011010101000100010010 1111
33	10	0	-1994937995	42553	11000011011011110000010100111 1011
34	1	0	3125629	49615	10001010010000000000000101000100 11
34	2	0	41858260	45262	10001000010000000010010101100000 101
34	3	0	592372480	53441	11110101110101100101001000001011 11
34	4	0	658271666	46089	11101010111000100100010000000111 101
34	5	0	548266115	57160	10110011000000101010001100000101 01
34	6	0	520566063	51150	11110000101000011111000000000100 101
34	7	0	672774409	54857	10110101010100100000000001100010 011
34	8	0	104686401	43925	11001110011110100010001101001010 11
34	9	0	1934462648	47333	11101011101100110000100000100101 11
34	10	0	-828159214	41142	01011001100111100001000101100000 11
35	1	0	10846352	59156	10011010000000000000000000000010 001
35	2	0	103715904	53119	01100001001000100011001001000100 101
35	3	0	643118458	40571	10110001110000111010000111011101000
35	4	0	1414846032	53295	11110110010010100011000010101001111
35	5	0	-1752920474	51159	11111001001000000000011000101101011
35	6	0	721569682	58376	10111000100011000000100000100110011
35	7	0	63321029	43489	11111100000001110100000001111011010
35	8	0	-942876697	60808	11111100010000100000100010111110010
35	9	0	139433788	36319	10111001100001110000011000101111111
35	10	0	-2125074731	49876	01111110010000100110010000010100111
36	1	0	8642100	52133	10100100010011000110000000000000000
36	2	0	49229109	42838	00110101000010000100000000100100101 0
36	3	0	1065581746	55184	01110001000001000011000100001000000 0

36	4	0	621953998	58511	11101000100000000000100000000000010
36	5	0	2114975280	46681	01100101100011101111110000010011111
36	6	0	-1948120579	52696	11111100011011000010000010011111000
36	7	0	1743987712	51056	11111110011011000010100000101100111
36	8	0	2096957496	50486	11101111001110100000010000100110011
36	9	0	-1971344239	51420	11111011001010000000110101010110010
36	10	0	533362018	47776	11111100100100110000010110001110011

Resultados dos Random Assignments

T	P	Best Profit (**Random Assignment**)	Best Profit **(Brute Force)**	Solution Time (**Random assignment**)	Solution Time **(Brute Force)**
1	1	4018	4018	1.003e+00	5.576e-05
2	1	4958	4958	2.025e+00	2.692e-05
2	2	3875	3875	2.115e+00	1.917e-05
3	1	2875	2875	3.183e+00	1.768e-05
3	2	2634	2634	3.211e+00	1.845e-05
3	3	4517	4517	3.143e+00	1.886e-05
4	1	4771	4771	4.358e+00	2.600e-05
4	2	6071	6071	4.135e+00	3.398e-05
4	3	7058	7058	4.114e+00	2.266e-05
4	4	10226	10226	4.153e+00	2.302e-05
5	1	3626	3626	5.200e+00	2.508e-05
5	2	7630	7630	5.182e+00	2.318e-05
5	3	7088	7088	5.160e+00	2.980e-05
5	4	9983	9983	5.233e+00	2.977e-05
5	5	10528	10528	5.191e+00	3.509e-05

6	1	4100	4100	6.287e+00	3.520e-05
6	2	8443	8443	6.234e+00	3.545e-05
6	3	12035	12035	6.233e+00	4.000e-05
6	4	10331	10331	6.189e+00	3.510e-05
6	5	11795	11795	6.167e+00	5.095e-05
6	6	9791	9791	6.260e+00	3.786e-05
7	1	9889	9889	7.233e+00	4.416e-05
7	2	6004	6004	7.242e+00	4.019e-05
7	3	11578	11578	7.237e+00	4.172e-05
7	4	9516	9516	7.304e+00	4.327e-05
7	5	9170	9170	7.320e+00	4.324e-05
7	6	14163	14163	7.411e+00	4.076e-05
7	7	16936	16936	7.567e+00	4.094e-05
8	1	7753	7753	8.524e+00	5.300e-05
8	2	8355	8355	8.467e+00	6.588e-05
8	3	7495	7495	8.250e+00	6.689e-05
8	4	12636	12636	8.175e+00	6.901e-05
8	5	12773	12773	8.297e+00	9.977e-05
8	6	19065	19065	8.236e+00	7.054e-05
8	7	16852	16852	8.194e+00	7.011e-05
8	8	14370	14370	7.989e+00	1.045e-04
9	1	11276	11276	8.941e+00	1.407e-04
9	2	9263	9263	9.030e+00	1.094e-04
9	3	8820	8820	9.448e+00	1.464e-04
9	4	13348	13348	9.203e+00	1.321e-04
9	5	20895	20895	9.177e+00	1.144e-04

9	6	12954	12954	9.225e+00	1.631e-04
9	7	13862	13862	9.487e+00	1.414e-04
9	8	16473	16473	9.322e+00	1.461e-04
9	9	21947	21947	9.148e+00	1.477e-04
10	1	18773	18773	1.022e+01	1.854e-04
10	2	12222	12222	1.043e+01	2.322e-04
10	3	14472	14472	1.029e+01	2.067e-04
10	4	14267	14267	1.044e+01	2.347e-04
10	5	12811	12811	1.030e+01	1.917e-04
10	6	17058	17058	1.014e+01	2.833e-04
10	7	19911	19911	1.013e+01	2.075e-04
10	8	17707	17707	1.011e+01	3.068e-04
10	9	21958	21958	1.006e+01	3.798e-04
10	10	28276	28276	1.038e+01	2.212e-04
11	1	19939	19939	1.124e+01	3.129e-04
11	2	13171	13171	1.202e+01	3.253e-04
11	3	14041	14041	1.123e+01	3.911e-04
11	4	15556	15556	1.161e+01	4.479e-04
11	5	18559	18559	1.132e+01	4.773e-04
11	6	19367	19367	1.165e+01	4.419e-04
11	7	11720	11720	1.100e+01	4.541e-04
11	8	17182	17182	1.124e+01	6.073e-04
11	9	19337	19337	1.139e+01	4.319e-04
11	10	25165	25165	1.136e+01	4.874e-04
12	1	13740	13740	1.256e+01	5.727e-04
12	2	14195	14195	1.233e+01	5.850e-04

12	3	13193	13193	1.233e+01	6.025e-04
12	4	15139	15139	1.270e+01	7.533e-04
12	5	18312	18312	1.225e+01	6.924e-04
12	6	21576	21576	1.233e+01	7.694e-04
12	7	21252	21252	1.189e+01	9.779e-04
12	8	22657	22657	1.205e+01	1.338e-03
12	9	25200	25200	1.219e+01	8.958e-04
12	10	21949	21949	1.202e+01	8.651e-04
13	1	16129	16129	1.287e+01	1.072e-03
13	2	9924	9924	1.295e+01	1.150e-03
13	3	20236	20236	1.290e+01	1.991e-03
13	4	19475	19475	1.322e+01	2.052e-03
13	5	17333	17333	1.398e+01	1.533e-03
13	6	17998	17998	1.382e+01	2.016e-03
13	7	20500	20500	1.389e+01	2.226e-03
13	8	20819	20819	1.389e+01	2.004e-03
13	9	26005	26005	1.387e+01	1.807e-03
13	10	26165	26165	1.412e+01	1.938e-03
14	1	13187	13187	1.497e+01	2.382e-03
14	2	15720	15720	1.485e+01	1.816e-03
14	3	21990	21990	1.486e+01	2.075e-03
14	4	16296	16296	1.486e+01	3.846e-03
14	5	17365	17365	1.486e+01	3.366e-03
14	6	26589	26589	1.485e+01	3.651e-03
14	7	24725	24725	1.484e+01	4.429e-03
14	8	27294	27294	1.489e+01	3.378e-03

14	9	29189	29226	1.484e+01	4.327e-03
14	10	30266	30266	1.482e+01	5.771e-03
15	1	19616	19616	1.581e+01	4.812e-03
15	2	16172	16906	1.594e+01	3.897e-03
15	3	18765	18780	1.588e+01	6.533e-03
15	4	20065	20842	1.590e+01	8.523e-03
15	5	15621	15621	1.576e+01	4.443e-03
15	6	20602	20716	1.591e+01	4.961e-03
15	7	25010	25720	1.589e+01	5.563e-03
15	8	28196	28764	1.583e+01	6.247e-03
15	9	23697	23895	1.594e+01	6.415e-03
15	10	27553	28229	1.588e+01	5.745e-03
16	1	16465	18246	1.703e+01	6.799e-03
16	2	17888	17888	1.688e+01	1.375e-02
16	3	17470	18231	1.691e+01	1.216e-02
16	4	16453	16714	1.689e+01	1.347e-02
16	5	30650	31576	1.682e+01	1.513e-02
16	6	19371	19371	1.681e+01	1.476e-02
16	7	23559	24270	1.713e+01	1.132e-02
16	8	31545	31656	1.694e+01	1.226e-02
16	9	37845	37845	1.686e+01	1.355e-02
16	10	25414	25888	1.701e+01	1.769e-02
17	1	29416	29416	1.796e+01	1.408e-02
17	2	24671	25110	1.800e+01	1.523e-02
17	3	18191	18832	1.789e+01	2.159e-02
17	4	24416	24459	1.814e+01	2.359e-02

17	5	27550	27604	1.814e+01	2.506e-02
17	6	22197	22487	1.804e+01	2.194e-02
17	7	30477	30485	1.810e+01	2.383e-02
17	8	31082	31082	1.806e+01	2.953e-02
17	9	27392	27496	1.809e+01	3.192e-02
17	10	25868	25868	1.809e+01	2.572e-02
18	1	25356	38215	1.902e+01	3.557e-02
18	2	23849	24939	1.898e+01	3.653e-02
18	3	25688	25773	1.894e+01	3.565e-02
18	4	25508	26389	1.895e+01	4.143e-02
18	5	23527	23971	1.895e+01	4.354e-02
18	6	22719	23383	1.896e+01	4.451e-02
18	7	25898	26539	1.897e+01	5.248e-02
18	8	27324	27564	1.905e+01	5.490e-02
18	9	30138	30138	1.894e+01	5.907e-02
18	10	30078	30078	1.889e+01	5.713e-02
19	1	29918	30117	1.993e+01	6.614e-02
19	2	22863	24167	1.998e+01	7.264e-02
19	3	26495	27660	1.994e+01	7.708e-02
19	4	25137	26392	1.993e+01	7.933e-02
19	5	28602	28602	2.002e+01	8.140e-02
19	6	24970	25363	1.998e+01	9.238e-02
19	7	25644	26657	2.004e+01	1.020e-01
19	8	27835	28400	2.011e+01	1.068e-01
19	9	32961	33769	2.017e+01	1.178e-01
19	10	30078	30568	2.017e+01	1.139e-01

20	1	20031	23617	2.105e+01	1.292e-01
20	2	30449	36597	2.107e+01	1.334e-01
20	3	26435	27672	2.105e+01	1.540e-01
20	4	20812	21561	2.110e+01	1.622e-01
20	5	22552	23809	2.099e+01	1.749e-01
20	6	25745	27458	2.097e+01	1.860e-01
20	7	29755	31608	2.102e+01	1.924e-01
20	8	24963	25943	2.092e+01	2.225e-01
20	9	24388	25163	2.109e+01	2.471e-01
20	10	36161	37617	2.106e+01	2.357e-01
21	1	20388	23870	2.229e+01	2.495e-01
21	2	28140	31382	2.248e+01	2.752e-01
21	3	30295	32060	2.256e+01	2.935e-01
21	4	26459	26823	2.229e+01	3.195e-01
21	5	26005	28038	2.263e+01	3.368e-01
21	6	23791	25030	2.225e+01	3.704e-01
21	7	25736	26445	2.219e+01	3.962e-01
21	8	30271	31694	2.232e+01	4.395e-01
21	9	28751	30359	2.231e+01	4.847e-01
21	10	33717	34948	2.236e+01	5.463e-01
22	1	24130	24130	2.330e+01	5.195e-01
22	2	26808	31358	2.321e+01	5.745e-01
22	3	28719	30352	2.322e+01	5.971e-01
22	4	35672	36391	2.328e+01	6.930e-01
22	5	25183	25611	2.320e+01	7.582e-01
22	6	24164	25028	2.322e+01	8.119e-01

22	7	23800	24728	2.342e+01	8.201e-01
22	8	30428	31584	2.321e+01	8.902e-01
22	9	25532	26602	2.345e+01	9.845e-01
22	10	38012	39154	2.327e+01	1.086e+00
23	1	17645	20814	2.445e+01	1.044e+00
23	2	31504	39677	2.444e+01	1.145e+00
23	3	41352	43207	2.444e+01	1.316e+00
23	4	36519	38674	2.430e+01	1.396e+00
23	5	26314	27877	2.434e+01	1.431e+00
23	6	32958	34285	2.445e+01	1.551e+00
23	7	29997	31934	2.451e+01	1.714e+00
23	8	31137	31698	2.463e+01	1.918e+00
23	9	34480	35435	2.440e+01	2.099e+00
23	10	30892	31679	2.437e+01	2.152e+00
24	1	22029	40687	2.534e+01	2.211e+00
24	2	26881	30337	2.531e+01	2.373e+00
24	3	32281	36206	2.528e+01	2.663e+00
24	4	35967	38088	2.531e+01	2.906e+00
24	5	25305	26699	2.535e+01	3.066e+00
24	6	36222	38406	2.518e+01	3.206e+00
24	7	34785	37833	2.533e+01	3.488e+00
24	8	25841	27055	2.525e+01	3.891e+00
24	9	30868	32816	2.538e+01	4.235e+00
24	10	28971	29560	2.547e+01	4.351e+00
25	1	19372	52643	2.628e+01	4.655e+00
25	2	25399	30361	2.633e+01	4.890e+00

25	3	22547	26762	2.656e+01	5.361e+00
25	4	33282	37499	2.628e+01	5.832e+00
25	5	31443	35819	2.620e+01	6.336e+00
25	6	30655	32398	2.616e+01	6.691e+00
25	7	33306	36293	2.628e+01	7.276e+00
25	8	36751	39602	2.689e+01	7.504e+00
25	9	31060	34310	2.748e+01	9.000e+00
25	10	35905	38090	2.637e+01	9.736e+00
26	1	21536	30742	2.724e+01	9.265e+00
26	2	28419	43764	2.739e+01	1.063e+01
26	3	36969	40906	2.761e+01	1.074e+01
26	4	36692	42004	2.755e+01	1.201e+01
26	5	37247	39712	2.789e+01	1.304e+01
26	6	37388	39317	2.788e+01	1.375e+01
26	7	31450	34944	2.750e+01	1.455e+01
26	8	36508	39360	2.753e+01	1.621e+01
26	9	35070	37918	2.728e+01	1.790e+01
26	10	35131	36857	2.738e+01	1.859e+01
27	1	19598	37757	2.829e+01	1.930e+01
27	2	28146	31216	2.834e+01	2.086e+01
27	3	33768	38810	2.827e+01	2.253e+01
27	4	36394	40590	2.828e+01	2.404e+01
27	5	34045	37187	2.826e+01	2.684e+01
27	6	35184	37633	2.833e+01	2.837e+01
27	7	26969	30287	2.847e+01	3.086e+01
27	8	35981	38734	2.831e+01	3.398e+01

27	9	34015	36579	2.830e+01	3.817e+01
27	10	37424	39166	2.849e+01	3.922e+01
28	1	23505	32437	2.949e+01	3.963e+01
28	2	31517	36272	3.010e+01	4.185e+01
28	3	39596	45002	3.042e+01	4.673e+01
28	4	33492	37461	3.013e+01	5.167e+01
28	5	34160	38691	2.980e+01	5.487e+01
28	6	34207	39418	2.987e+01	5.866e+01
28	7	34660	36970	2.977e+01	6.294e+01
28	8	30950	32532	2.989e+01	7.149e+01
28	9	38180	41687	3.054e+01	7.841e+01
28	10	37454	41476	3.038e+01	8.139e+01
29	1	20220	47245	3.133e+01	8.162e+01
29	2	30755	36831	3.129e+01	8.637e+01
29	3	40455	44922	3.056e+01	9.635e+01
29	4	29382	34613	3.071e+01	1.040e+02
29	5	47379	51126	3.055e+01	1.157e+02
29	6	44647	46038	3.056e+01	1.188e+02
29	7	40948	46140	3.090e+01	1.300e+02
29	8	32837	36454	3.041e+01	1.409e+02
29	9	34579	37932	3.043e+01	1.587e+02
29	10	41182	42941	3.051e+01	1.762e+02
30	1	11790	36741	3.132e+01	1.688e+02
30	2	32671	41011	3.135e+01	1.783e+02
30	3	37519	42384	3.160e+01	1.947e+02
30	4	43933	47985	3.138e+01	2.131e+02

30	5	42947	51329	3.142e+01	2.360e+02
30	6	41619	47955	3.148e+01	2.488e+02
30	7	34179	38097	3.146e+01	2.703e+02
30	8	39632	45837	3.148e+01	2.923e+02
30	9	38665	41398	3.148e+01	3.238e+02
30	10	39997	42820	3.142e+01	3.643e+02
31	1	23531	33793	3.244e+01	3.395e+02
31	2	31512	39237	3.247e+01	3.785e+02
31	3	37205	52470	3.266e+01	4.098e+02
31	4	42562	47737	3.275e+01	4.363e+02
31	5	36659	43335	3.268e+01	5.098e+02
31	6	43990	49555	3.275e+01	5.026e+02
31	7	39946	44164	3.258e+01	5.479e+02
31	8	38186	40808	3.250e+01	6.434e+02
31	9	35620	38837	3.250e+01	6.673e+02
31	10	35453	39029	3.251e+01	8.301e+02
32	1	29123	42307	3.352e+01	7.030e+02
32	2	34671	44666	3.377e+01	7.558e+02
32	3	34701	41847	3.388e+01	8.129e+02
32	4	46180	50484	3.387e+01	9.007e+02
32	5	32734	37878	3.376e+01	9.879e+02
32	6	41371	46567	3.371e+01	1.028e+03
32	7	37474	39889	3.391e+01	1.158e+03
32	8	38587	42146	3.367e+01	1.253e+03
32	9	40550	43814	3.366e+01	1.420e+03
32	10	42654	46911	3.388e+01	1.571e+03

33	1	20249	34304	3.451e+01	1.462e+03
33	2	30596	47770	3.464e+01	1.682e+03
33	3	39317	50803	3.445e+01	1.707e+03
33	4	46086	52204	3.463e+01	1.856e+03
33	5	49284	55356	3.459e+01	2.093e+03
33	6	45604	48753	3.474e+01	2.163e+03
33	7	48497	54161	3.450e+01	2.260e+03
33	8	42996	46456	3.468e+01	2.540e+03
33	9	43413	46058	3.496e+01	2.866e+03
33	10	38938	42553	3.521e+01	3.474e+03
34	1	31603	49615	3.654e+01	2.983e+03
34	2	38011	45262	3.638e+01	3.236e+03
34	3	43460	53441	3.607e+01	3.315e+03
34	4	41290	46089	3.631e+01	3.684e+03
34	5	48997	57160	3.647e+01	4.225e+03
34	6	43207	51150	3.626e+01	4.496e+03
34	7	48453	54857	3.667e+01	5.187e+03
34	8	39435	43925	3.660e+01	5.204e+03
34	9	41374	47333	3.624e+01	6.003e+03
34	10	38651	41142	3.617e+01	7.348e+03
35	1	18872	59156	3.748e+01	6.126e+03
35	2	25916	53119	3.734e+01	6.519e+03
35	3	31350	40571	3.675e+01	7.076e+03
35	4	40994	53295	3.687e+01	7.496e+03
35	5	43173	51159	3.700e+01	8.421e+03
35	6	49133	58376	3.823e+01	1.020e+04

35	7	41046	43489	3.752e+01	1.006e+04
35	8	53145	60808	3.751e+01	1.152e+04
35	9	32277	36319	3.744e+01	1.261e+04
35	10	41743	49876	3.769e+01	1.452e+04
36	1	23529	52133	3.874e+01	1.236e+04
36	2	26510	42838	3.876e+01	1.390e+04
36	3	40055	55184	3.830e+01	1.500e+04
36	4	45934	58511	3.833e+01	1.677e+04
36	5	42490	46681	3.828e+01	1.657e+04
36	6	45884	52696	3.828e+01	1.862e+04
36	7	43878	51056	3.831e+01	1.947e+04
36	8	45624	50486	3.871e+01	2.262e+04
36	9	44841	51420	3.868e+01	2.545e+04
36	10	43768	47776	3.875e+01	2.920e+04