Algoritmo e Estrutura de Dados

Licenciatura em Engenharia Informática

Generalized Weighted Job Selection Problem

Professor: Tomás Oliveira e Silva.

Data: 05/01/2021

Trabalho realizado por:

Daniel Figueiredo, nº 98498, 33,33%

Eva Bartolomeu, nº 98513, 33,33%

Eduardo Fernandes, nº 98512, 33,33%

Índice

Índice

1. Introdução

No âmbito da cadeira de Algoritmos e Estruturas de Dados, foi-nos proposto um trabalho acerca de Generalized weighted job selection problem.

Este problema baseia-se num conjunto de tarefas e programadores. Uma tarefa é caracterizada pelo lucro, e pelas datas de começo e fim.

A cada tarefa associa-se, ou não, um programador, criando-se assim um conjunto de tarefas relacionadas com os programadores. Mas este conjunto de associações tarefa-programador, só é válido se seguir algumas regras:

- Uma tarefa só pode estar associada a um programador;
- Um programador só pode realizar uma tarefa de cada vez;
- Um programador quando começa uma tarefa é obrigado a terminá-la.

A solução deste problema é encontrar o subconjunto de tarefas que possui o lucro total maior, considerando-se assim este subconjunto o melhor de todos. O lucro total de um conjunto é a soma dos lucros das tarefas que foram realizadas.

Os objetivos deste relatório são:

- Desenvolver um código C capaz de resolver o problema proposto;
- Adaptação do número de tarefas à função implementada;
- Obter um bom tempo de execução para as tarefas realizadas;
- Elaboração de tabelas dos melhores lucros perante o número de tarefas e programadores;
- Interpretação das tabelas;
- Construção de gráficos que relacionam o número de tarefas, número de programadores e tempos de execução;
- Construção de gráficos que relacionam o número de tarefas, número de programadores e lucros;
- Capacidade de interpretar e de descobrir a razão dessa mesma interpretação;
- Encontrar o número de tarefas válidas;
- Construção e interpretação de histogramas dos melhores lucros totais;
- Construção e interpretação de histogramas do número máximo de tarefas;
- Construção e interpretação de gráficos que relacionam o número de soluções e o número máximo de tarefas possíveis.

2. Método de resolução

No início deste projeto fomos confrontados com 2 métodos possíveis para a resolução deste trabalho prático. O primeiro seria a implementação de código capaz de calcular todas as maneiras possíveis de distribuir as tarefas pelos programadores, verificando se estas eram viáveis e, de seguida, verificar qual delas possuía maior lucro. A segunda implementação seria uma função recursiva capaz de calcular as tarefas que concretizam maior lucro.

A primeira implementação, embora funcione, coloca-nos numa posição onde todas as possibilidades teriam de ser calculadas independentemente se já tivéssemos encontrado o nosso objetivo ou não. Isto apesar de funcionar é um método onde o tempo de execução seria sempre o mais demorado possível, aspeto que devemos evitar.

Foi então que decidimos optar por implementar então a função recursiva (Fig. 1), que permite um tempo de execução menor relativamente à primeira ideia de resolução do problema.

A função construída foi a seguinte:

```
275  void best_subset(problem_t *problem, int num_tarefas)
276  {
277
278    int prog;
279
280    if (num_tarefas == problem->T)
281    {                                                    //Calculo do profit
282      problem->num_viables++;                            // tasks viaveis ++
283      if (problem->total_profit > problem->best_total_profit)
284      {
285        problem->best_total_profit = problem->total_profit;
286        problem->num_solutions = 1;
287
288        for (int j = 0; j < problem->T; j++)
289        {
290          problem->task[j].best_assigned_to = problem->task[j].assigned_to;
291        }
292      }
293      else if (problem->total_profit == problem->best_total_profit)
294      {
295        problem->num_solutions++;
296      }
297
298    }
299    else
300    {
301      problem->task[num_tarefas].assigned_to = -1;
302      best_subset(problem, num_tarefas + 1);
303
304      for (prog = 0; prog < problem->P; prog++)
305      {
306        if (problem->task[num_tarefas].starting_date > problem->busy[prog])
307        {
308          break;
309        }
310      }
311
312      if (prog < problem->P)
313      {
314
315        int profit_tmp = problem->total_profit;
316        int busy_tmp = problem->busy[prog];
317
318        problem->busy[prog] = problem->task[num_tarefas].ending_date;
319        problem->total_profit += problem->task[num_tarefas].profit;
320        problem->task[num_tarefas].assigned_to = prog;
321
322        best_subset(problem, num_tarefas + 1);
323
324        problem->total_profit = profit_tmp;
325        problem->busy[prog] = busy_tmp;
326      }
327    }
328  }
```

Figura 1: Código C da função recursiva 'best_subset'

Esta função tem como nome *best_subset*, tendo como argumentos de entrada uma variável *problem,* e uma variável *num_tarefas* que vai sendo alterada à medida que a função é chamada recursivamente.

A função começa por inicializar uma variável *prog* que será necessária posteriormente para calcular o índice de um programador sob uma determinada condição e, de seguida, entra na parte principal da função.

É nesta parte onde é verificado se a variável *num_tarefas* é igual ao número de tarefas do problema. Caso não seja, é executado as linhas de código onde a função é chamada recursivamente, que são as seguintes:

```
299     else
300     {
301       problem->task[num_tarefas].assigned_to = -1;
302       best_subset(problem, num_tarefas + 1);
303
304       for (prog = 0; prog < problem->P; prog++)
305       {
306         if (problem->task[num_tarefas].starting_date > problem->busy[prog])
307         {
308           break;
309         }
310       }
311
312       if (prog < problem->P)
313       {
314
315         int profit_tmp = problem->total_profit;
316         int busy_tmp = problem->busy[prog];
317
318         problem->busy[prog] = problem->task[num_tarefas].ending_date;
319         problem->total_profit += problem->task[num_tarefas].profit;
320         problem->task[num_tarefas].assigned_to = prog;
321
322         best_subset(problem, num_tarefas + 1);
323
324         problem->total_profit = profit_tmp;
325         problem->busy[prog] = busy_tmp;
326       }
327     }
328   }
```

Figura 2: Parte do código C da função 'best_subset'

Como é nesta parte onde a função é chamada recursivamente é necessário fazer alterações tanto relativamente às tarefas como aos programadores.

Antes da função ser chamada pela primeira vez, é definido o valor da variável *assigned_to* da tarefa em questão para -1, indicando que esta tarefa não está associada a nenhum programador.

Na linha 304 é percorrido um ciclo *for* no qual o objetivo é encontrar um programador que esteja livre para executar uma determinada tarefa. Isto é possível verificando se a *starting_date* da tarefa é superior ao valor contido no vetor *busy* (vetor que possui valores que são interpretados de forma a saber se um programador está ocupado ou não).

Neste vetor pode ser encontrado o valor -1, que indica que um programador está livre, ou valores inteiros positivos, que indicam a *ending_date* de uma determinada tarefa já executada pelo programador. Caso o valor da *starting_date* seja maior que o valor contido no *busy* para um

determinado programador então, o ciclo é interrompido e ficamos com o índice do programador que está apto a executar a tarefa proposta.

Seguidamente, é verificado se o índice do programador é inferior ao número de programadores entrando na área onde vão ser atualizados alguns valores do problema.

```
315         int profit_tmp = problem->total_profit;
316         int busy_tmp = problem->busy[prog];
317
318         problem->busy[prog] = problem->task[num_tarefas].ending_date;
319         problem->total_profit += problem->task[num_tarefas].profit;
320         problem->task[num_tarefas].assigned_to = prog;
321
322         best_subset(problem, num_tarefas + 1);
323
324         problem->total_profit = profit_tmp;
325         problem->busy[prog] = busy_tmp;
```

Figura 3: Atualização de alguns valores

Inicialmente, colocamos os valores do *total_profit* e do *busy[prog]* em variáveis temporárias na medida em que, estas irão ser alteradas. Tendo sido guardados estes valores, o valor da variável *busy[prog]* é alterado para a *ending_date* da tarefa, de forma a indicar que este determinado programador estará ocupado até à data definida. Para além disso, é somado à variável *total_profit* o lucro da tarefa em questão sendo também, o *assigned_to* da tarefa alterado para o índice do respetivo programador que a realizará.

Após serem feitas estas alterações, podemos então chamar novamente a função recursivamente, aumentando o argumento de entrada *num_tarefas* em 1 valor.

Podemos então novamente retornar os valores que foram guardados nas variáveis temporárias para as originais.

Ao chamar a função recursivamente o valor do *num_tarefas* vai aumentando e quando este é igual ao número de tarefas do problema é executado a outra parte, já referida, do código da função.

```
280     if (num_tarefas == problem->T)
281     {                                                    //Calculo do pro
282       problem->num_viables++;                            // tasks viaveis
283       if (problem->total_profit > problem->best_total_profit)
284       {
285         problem->best_total_profit = problem->total_profit;
286         problem->num_solutions = 1;
287
288         for (int j = 0; j < problem->T; j++)
289         {
290           problem->task[j].best_assigned_to = problem->task[j].assigned_to;
291         }
292       }
293       else if (problem->total_profit == problem->best_total_profit)
294       {
295         problem->num_solutions++;
296       }
297
298     }
```

Figura 4: Parte do código C da função ''

Inicialmente, quando esta condição é verdadeira, acresce-se o valor, em uma unidade, da variável definida para o número de trabalhos viáveis, *num_viables*.

Após aumentar o valor de *num_viables*, é verificada uma condição. Caso o lucro total (*total_profit*) seja superior ao melhor lucro total (*best_total_profit*), altera-se o valor do *best_total_profit* igualando-o ao valor do *total_profit*, garantindo assim que o melhor lucro total seja guardado na variável correta. Para além disso, o número de soluções do problema é alterado para o valor 1 pois, neste caso, o lucro não é ignorado sendo que só existe uma solução possível com o lucro máximo para o problema. É ainda executado um ciclo *for* que permite percorrer todas as tarefas do problema passando o valor do *assigned_to* de cada tarefa para o *best_assigned_to* uma vez que encontramos a melhor solução para o problema.

Caso o lucro total seja igual ao melhor lucro possível, acrescenta-se em 1 valor o número de soluções possíveis. Isto acontece quando o lucro é ignorado.

```
351     // call your (recursive?) function to solve the problem here
352     problem->num_viables = 0;
353     problem->total_profit = 0;
354     problem->best_total_profit = 0;
355
356     for (int j = 0; j < problem->P; j++)
357     {
358       problem->busy[j] = -1;
359     }
360
361   best_subset(problem, 0);
```

Figura 5: Inicialização de variáveis

Antes da função ser chamada pela primeira vez, temos de igualar o valor de algumas variáveis a 0. Neste caso, o número de tarefas viáveis (*num_viables*), o lucro total (*total_profit*) e o melhor lucro possível (*best_total_profit*) necessitam de ser 0 antes de ser chamada a função.

Além disto, é necessário percorrer os programadores do problema para lhes atribuir o valor -1 no vetor *busy*, de maneira que significa que cada um destes programadores se encontra livre para executar uma e qualquer tarefa disponível.

3. Resultados

Através do script “job_selection_do_all.sh” (Pág. 28, Fig. 6), conseguimos correr o código compilado atrás descrito com três argumentos. O primeiro vai ter valores de 1 a 39, representando o número de tarefas, o segundo vai de 1 a 8, correspondendo ao número de programadores e o último vai ser o valor de I, que indica se o lucro vai ser ignorado ou não.

Corremos esse código com cada um dos nossos números mecanográficos, e com a variável I a tomar valores de 0 e 1, criando assim 3 diretorias com vários ficheiros. Cada ficheiro é um problema com um determinado número de tarefas e programadores, contendo várias informações úteis para futura análise.

A principal resposta do problema está nos ficheiros que acabam em “_0.txt”, nas linhas que indicam o “Best total profit”, que é o valor do maior lucro possível para os determinados dados que estão lá.

Decidimos apresentar apenas as soluções, em que as tarefas têm os valores de 1 a 39. Pois tanto para I = 0 como para I = 1, verificamos que o tempo de execução para um número de tarefas 40 já era muito longo.

3.1 – Tabela dos melhores lucros totais

De modo a que, para cada número mecanográfico se mostre uma tabela dos melhores lucros, decidimos criar um script "best_profits.sh" (Pag. 28, Fig. 7). Este script vai criar para cada diretoria um ficheiro de nome "best_profits.txt", em que cada linha vai ter primeiramente o número de tarefas de um determinado problema, depois o número de programadores, e os melhores lucros de todos os subconjuntos, tarefas-programadores válidos.

Ainda para ajudar na construção das tabelas, implementamos um código Matlab no ficheiro "best_profits.m", visível na Figura 8 (Pág. 28) e na Figura 10 (Pág. 29), secção "Código usado para guardar informação útil para a construção de tabelas dos melhores lucros em word". **Nesta secção vai se criar as variáveis "tabela_98498", "tabela_98513" e "tabela_98512", através do ficheiro "best_profits.txt", e é com estas variáveis que formamos a Tabela 1, Tabela 2 e Tabela 3.**

Tasks / Programadores	1	2	3	4	5	6	7	8
1	1669							
2	2300	4649						
3	2883	5587	7437					
4	3674	6476	6623	8534				
5	6953	6398	10446	5826	11893			
6	9676	6894	12100	9553	12863	15581		
7	5140	8096	11157	12812	11469	13641	15690	
8	10664	6765	8928	16061	15199	14465	15894	24110
9	9919	11609	11697	13246	18131	11585	20471	16754
10	13037	10641	16962	14905	20065	20122	24381	21105
11	11925	16896	9276	11146	17307	13576	21047	16917
12	10635	15617	15005	13931	16458	17056	23342	18740
13	10365	18812	18042	15248	15916	21366	16637	19657
14	24471	16970	19100	15399	18627	16717	23748	21771
15	16196	21906	14420	20231	18889	25979	21878	25539
16	16719	18343	17299	18305	22784	25602	24213	24216
17	23791	22997	27959	22183	18484	26589	28881	29066
18	25206	22572	21938	26155	26445	21827	26706	25717
19	19033	25112	29007	26345	22821	21872	26092	20474
20	20997	22466	34869	28223	32041	27287	25908	29756
21	29566	38776	26637	28208	31204	25248	27030	30935
22	26346	29027	25378	30626	28394	23195	26494	29840
23	28071	30406	46269	34608	38815	32181	29226	28779
24	32082	33854	39986	29452	34073	27724	27907	28536
25	26447	32975	36663	33524	31762	34952	32568	32044
26	30373	35405	34874	35633	41276	26968	35682	30822
27	36643	42288	45367	39503	42341	32810	28753	35012
28	44681	32600	42400	36168	47047	38847	37231	35168
29	28504	35662	44052	50913	41688	36798	36309	40081
30	56242	44145	35879	45097	40737	41819	40769	36439
31	61082	33045	40952	45836	54200	43739	32927	44663
32	43563	48581	48098	43288	44359	42711	37322	44253
33	43432	66242	54917	40541	41347	45733	45283	39234
34	36815	42550	48831	53640	47524	47379	48439	46562
35	43743	39590	50784	44453	40751	45749	45810	49215
36	54202	54830	52634	43855	55822	53708	54674	50268
37	39990	80408	43158	51859	48663	59939	51296	62873
38	47745	48364	57687	60488	56087	61563	46451	54890
39	62304	54167	56368	50457	53377	62611	58441	46854

Tabela 1: Tabela dos melhores lucros, NMec = 98512.

Tasks / Programadores	1	2	3	4	5	6	7	8
1	998							
2	3145	3254						
3	1637	4349	7633					
4	2746	5016	5497	4909				
5	4183	8593	6821	11173	10366			
6	5427	8434	10336	14418	15159	10426		
7	7064	13084	14319	12882	14876	8744	12792	
8	7936	8399	8521	12319	17020	19090	18054	17607
9	9119	8329	13115	15240	15296	21562	14561	16800
10	10285	11315	13090	12858	16476	19489	16480	17311
11	9708	12324	13213	16788	16065	21488	18135	20385
12	13823	16981	18035	16423	17064	23741	23838	19653
13	12159	13885	18213	15052	17754	17819	21758	20541
14	13613	19167	13308	19276	15550	22656	19649	26646
15	14630	22171	22992	19142	16083	19536	17439	28718
16	17631	22331	25101	18733	21727	22895	23284	31477
17	25625	20147	25594	23968	19500	22259	25294	31511
18	19745	25433	19426	27696	28034	23102	21868	30452
19	26125	35207	27956	27643	28506	23708	24321	28095
20	18954	29516	29204	30902	28801	27924	31698	34234
21	25661	34491	35320	28459	26818	30584	29384	34730
22	39555	24769	32868	34084	24231	28509	32069	24400
23	48952	31992	35339	29704	30214	24514	28337	32435
24	33671	34134	33004	25459	33248	30008	31247	32808
25	57419	52443	34495	36527	37957	36766	32250	34371
26	55615	34931	37880	28057	26289	34231	36209	35137
27	40042	35946	44688	42697	34521	32332	37342	33583
28	33861	30045	47180	46643	37246	41601	36127	38018
29	44982	38768	39904	41677	44336	44946	46793	42391
30	23239	52699	39412	40787	37871	36641	38553	39935
31	36732	43030	45537	41828	46114	51576	44447	36213
32	44981	62855	37323	44100	41564	43163	51134	44772
33	37611	59538	44550	55579	53646	54102	47126	42852
34	33319	47945	53228	43937	44338	55748	55743	45428
35	61082	38815	43432	51536	48385	60738	39691	51637
36	38636	55125	48963	59012	51417	60832	58463	52280
37	55363	56029	64203	55179	48749	47567	52904	62071
38	56509	56636	52633	43061	52021	50575	57411	58332
39	50710	42781	73416	59162	54711	66444	60662	52100

Tabela 2: Tabela dos melhores lucros, NMec = 98498.

Tasks / Programadores	1	2	3	4	5	6	7	8
1	816							
2	1422	3474						
3	3927	5792	7055					
4	4571	9170	3762	8250				
5	4313	8448	9291	8912	16929			
6	7122	5215	8363	9392	10743	14864		
7	8234	7893	8845	12770	13042	12202	9661	
8	9777	6848	11306	11880	14823	14772	23188	17762
9	11448	9140	10979	15145	12917	19948	16953	21446
10	11588	15232	11381	12101	14087	20286	14894	19224
11	12476	12762	17055	12898	17506	21166	18818	22963
12	11447	20499	14713	12683	13164	21080	22636	19318
13	13458	14291	12696	18318	17105	18512	27206	28732
14	18072	21044	19172	13279	17823	20273	28416	25957
15	21465	19274	18539	23746	18444	22073	26064	24077
16	16483	20239	19047	18330	21543	23281	19303	24273
17	20015	20235	25765	21072	17915	22919	23435	29221
18	22975	26463	22297	26161	19784	23495	25303	26191
19	30468	33728	21797	20293	21087	23754	24757	30443
20	27150	28705	23862	23435	28269	20551	27370	31554
21	23360	37860	32083	29278	31568	26081	26224	32434
22	26190	36391	26629	23593	24136	24955	24921	35322
23	18703	27439	25131	33363	38275	27712	29875	26117
24	22525	27141	40886	26992	26672	32449	40086	30024
25	24209	27514	28975	33521	33193	35999	32215	30718
26	50678	43008	25145	33085	37605	36375	37242	40761
27	33794	41613	35594	36677	48139	39303	35131	35052
28	26807	33651	45686	41980	40005	43558	35062	38929
29	30932	52711	47607	44057	42768	41166	50201	35017
30	52558	44092	40464	34075	53225	41354	43581	41039
31	35562	36968	42373	49817	48999	59065	39217	45322
32	51959	50711	32685	46242	45380	48601	40493	41747
33	48122	46246	44629	57141	48497	56901	41167	52765
34	33271	60266	53225	46517	48119	45204	50242	43404
35	73772	64809	61498	47328	45438	46514	49276	49825
36	52966	52641	47300	52139	53531	58935	57547	46187
37	43385	80013	57707	62324	52543	48066	57113	55614
38	64148	67340	48391	52273	55311	66200	58541	59220
39	67215	71394	47648	62946	62983	53464	51440	60973

Tabela 3: Tabela dos melhores lucros, NMec = 98513.

A descrição destas tabelas, e as razões pelas quais são encontradas dessa maneira, estão descritas mais à frente na Pág. 17, com o Gráfico 3 a forçar a ideia.

3.2 – Tempos de execução

Para apresentar o gráfico de tempos de execução, usamos o script fornecido pelo professor “extract_data.sh” (Pág.30 Fig. 12). Executamos este script com os três números mecanográficos e adequamos o nome do ficheiro criado neste script a estes números. Criamos assim, três ficheiros com os tempos de execução, um para cada número mecanográfico.

Através do código matlab do ficheiro “graficos_tempos.m”, na secção “Gráfico dos tempos de execução para I=0” (Pág 30 Fig. 13/Pág 31 Fig. 14) e dos ficheiros criados anteriormente, construímos um gráfico (Gráfico 1). Cujos eixos irão representar o número de tarefas, programadores e os tempos de execução tal como é ilustrado. O vermelho simboliza o número mecanográfico 98513, o verde o número 98498, e o azul o 98512.

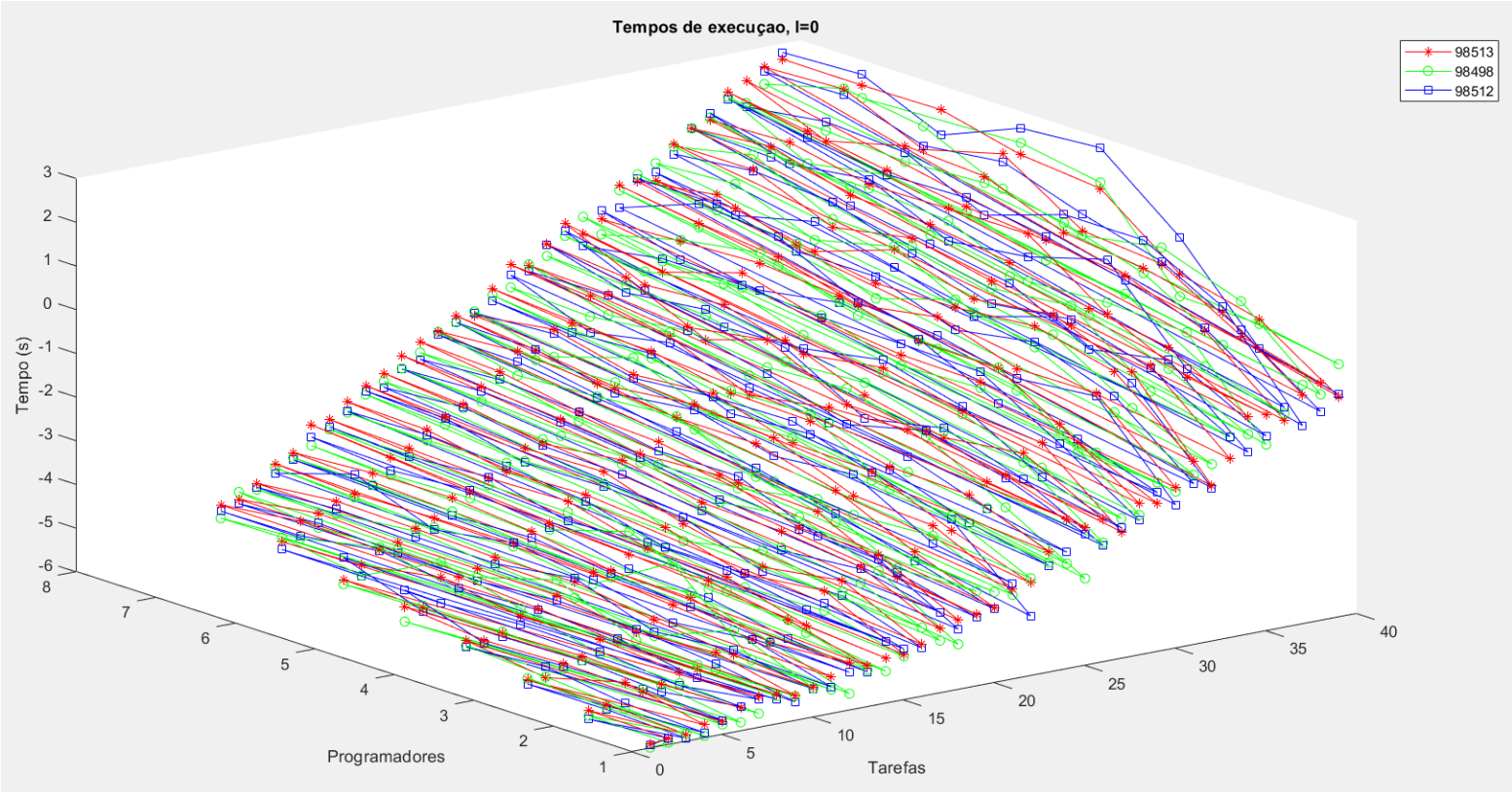


Gráfico 1: Tempos de execução da best_subset, com I = 0.

Neste gráfico é visível que quanto maior for o número de tarefas e o número de programadores, maior será o tempo de execução da "best_subset" (função implementada no ficheiro "job_selection.c").

Uma possível razão que justifique esta relação é pelo facto, de quanto maior o número de programadores, ou tarefas, mais subconjuntos de tarefas-programadores podem existir, ou seja, mais subconjuntos para validar e para analisar o lucro, de modo a descobrir a solução do problema (subconjunto com o lucro maior).

Apesar de não ser pedido, decidimos também fazer um gráfico de tempos de execução quando o lucro é ignorado. Utilizando, tal como no outro gráfico, os ficheiros "t1_98498", "t1_98512" e "t1_98513" e um código matlab expresso em "graficos_tempos.m", na secção "Gráfico dos tempos de execução para I=1" (Pág 31 Fig. 15). Os eixos e as cores deste gráfico têm o mesmo significado que o outro, como se pode ver no Gráfico 2 abaixo apresentado.

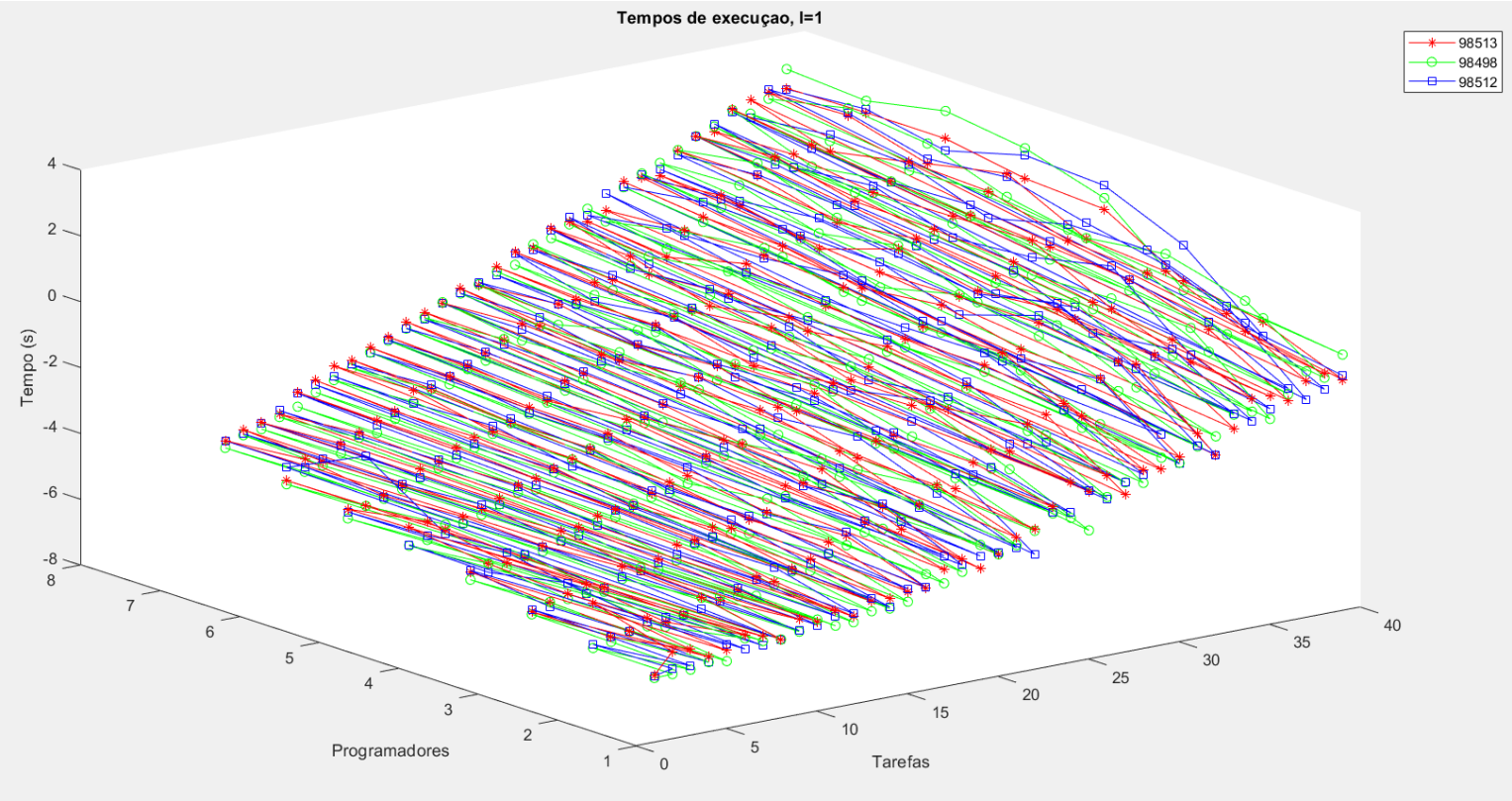


Gráfico 2: Tempos de execução da best_subset, com I = 1.

Tal como no gráfico anterior podemos verificar que quanto maior for o número de tarefas e o número de programadores, maior será o tempo de execução da "best_subset". Pois, há mais subconjuntos de tarefas-programadores, e por isso, há mais subconjuntos para validar e para analisar o número de tarefas realizadas, tornando o tempo mais comprido.

3.3 – Gráfico melhores lucros totais

De forma a desenhar as soluções para o N maior que conseguimos, resolvemos desenvolver um gráfico de três eixos que irão representar o número de tarefas, programadores e os melhores lucros. Para a elaboração deste gráfico utilizamos o código Matlab definido no ficheiro "best_profits.m" (Pág. 30 Fig. 11), na secção "Gráficos dos melhores lucros". Desenvolvendo-se assim o Gráfico 3, em que as cores representam os mesmos números mecanográficos mencionados anteriormente.

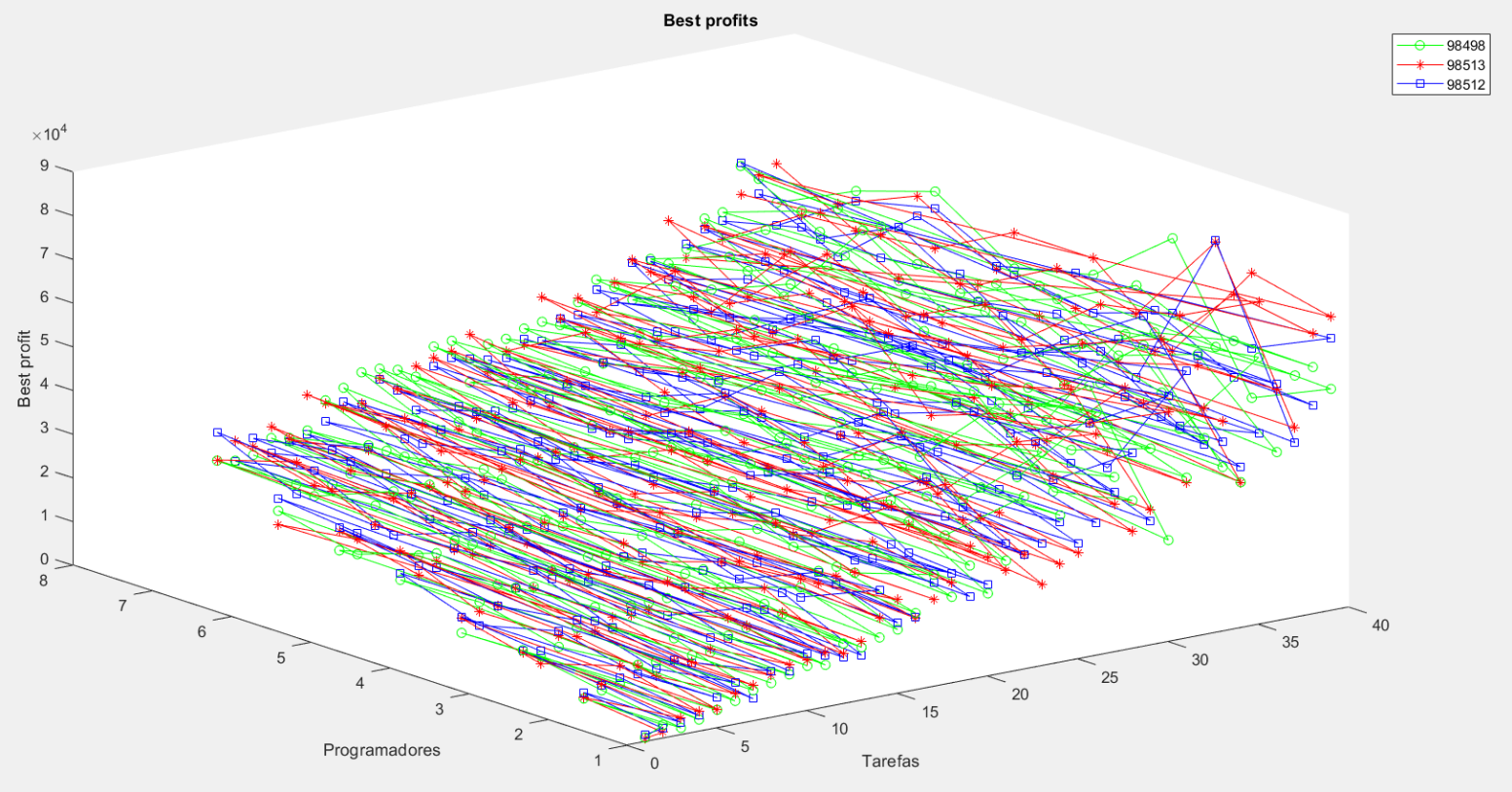


Gráfico 3: Melhores lucros.

No Gráfico 3, que não é nada mais nada menos do que a representação da Tabela 1, Tabela 2 e Tabela 3 num gráfico, podemos verificar que geralmente quanto maior o número de tarefas e programadores, maior serão os melhores lucros. Isto porque, quando há mais tarefas há mais chances de o lucro total ser maior, e quando há mais programadores poderão se realizar mais tarefas, logo o lucro total também será maior.

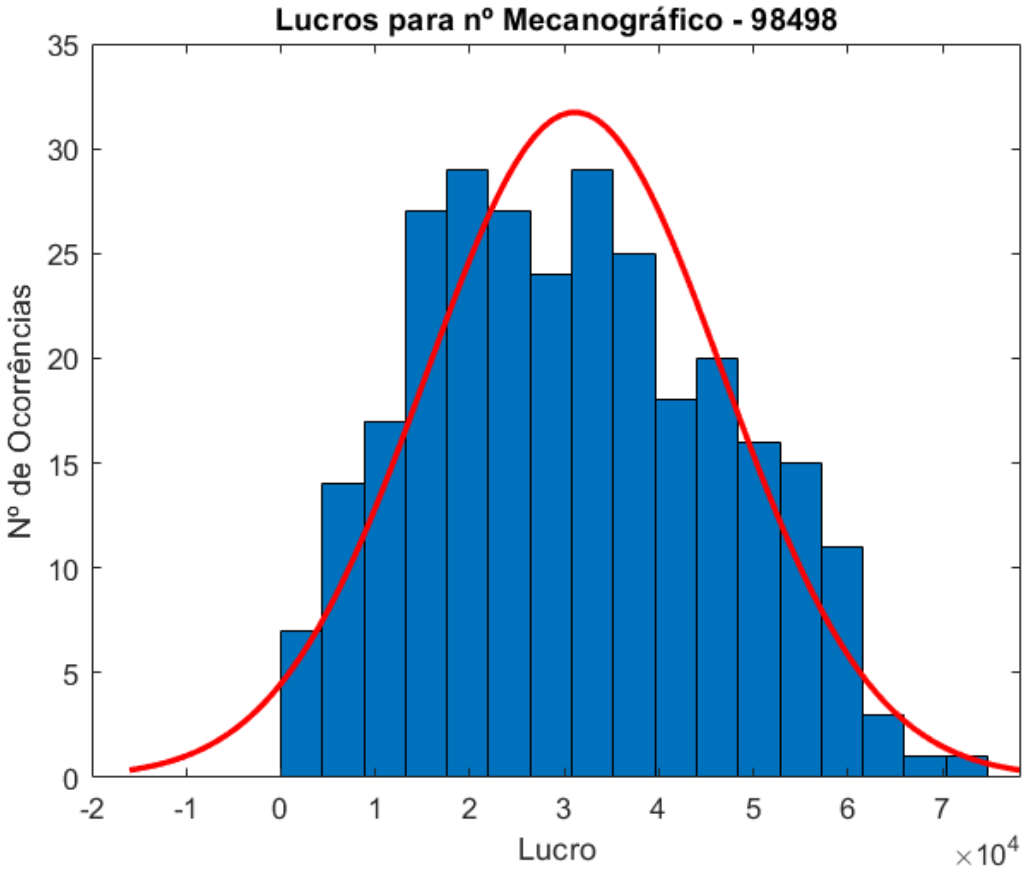
3.4 – Número de tarefas válidas

Com intuito de contar o número de tarefas válidas, criamos o script "total_valid_tasks.bash" (Pág. 32 Fig.16), onde é apresentado no output o total de tarefas válidas do número mecanográfico passado para o script. Executamos este código para os três números mecanográficos verificando -se para o número 98498 o valor de 497905464659, para 98512 o valor de 491381213805 e para o 98513 de 370367746966.

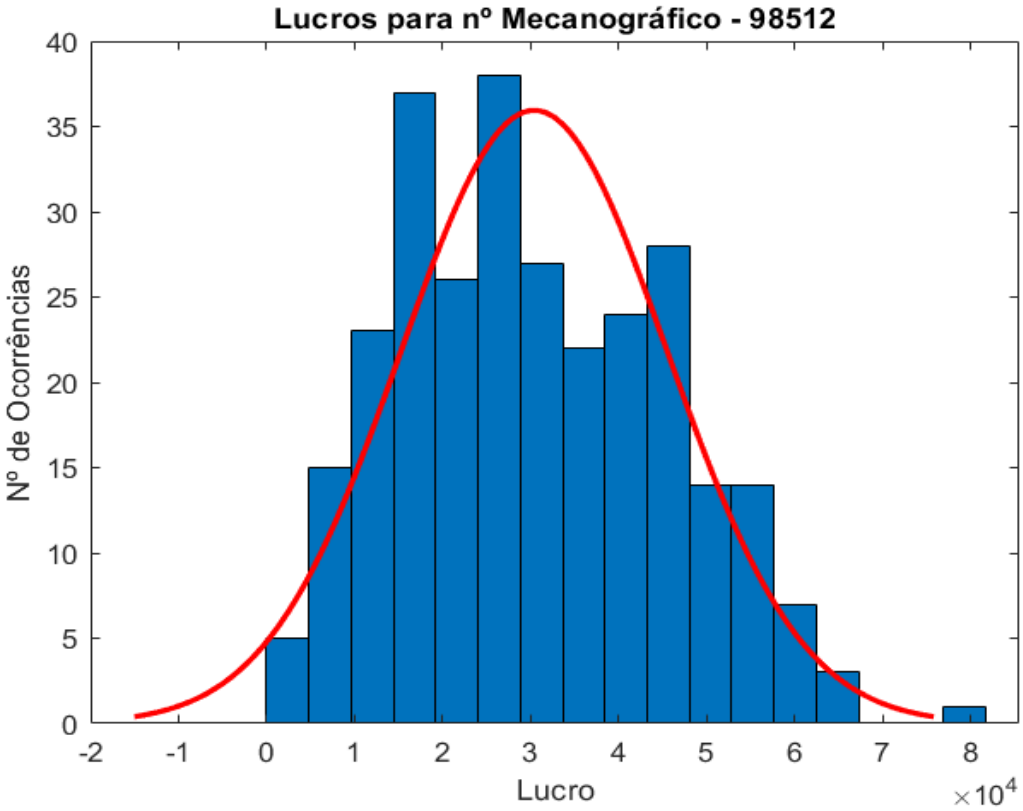
3.5 – Histogramas melhores lucros totais

Para os ficheiros criados com o script “job_selection_do_all.sh” (Pág 28 Fig. 6), ao utilizarmos para a variável I o valor 0, vamos obter ficheiros com informações diversas, como por exemplo, o tempo de solução, possibilidades para escolha de tarefas, melhor lucro obtido e propriedades de cada tarefa.

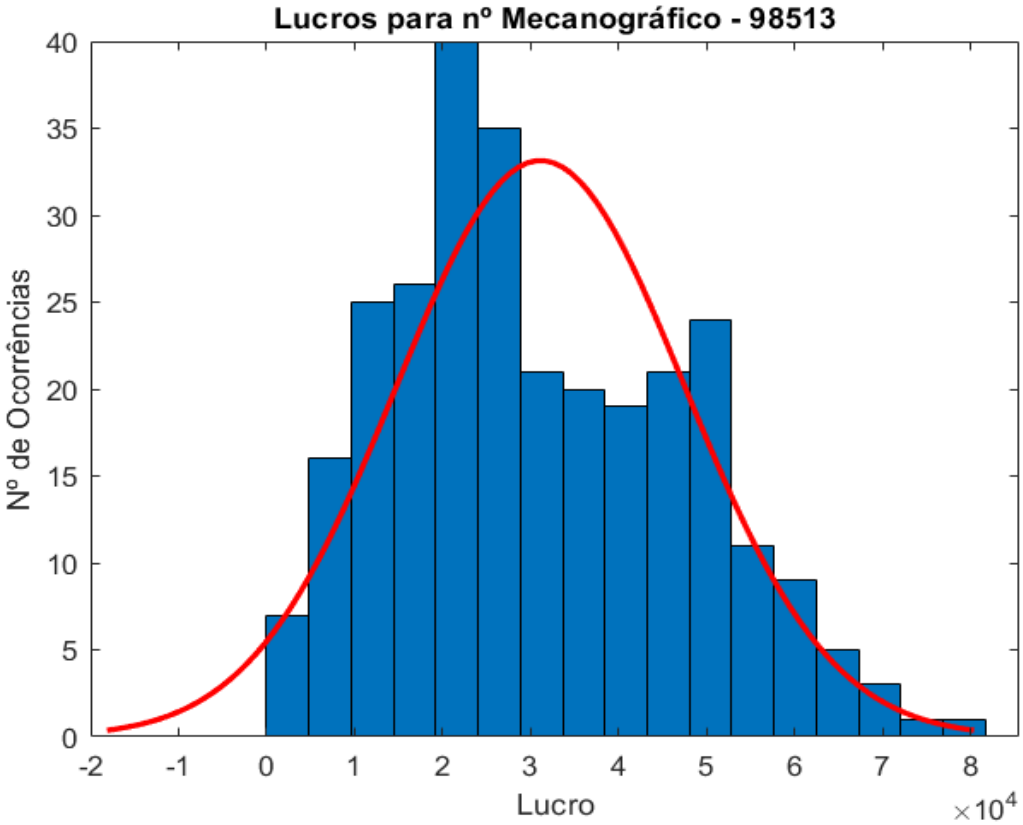
Para criarmos os histogramas dos melhores lucros, utilizamos o ficheiro “best_profits.m” (Pág 29 Fig. 9), implementado em Matlab e para obter os valores que foram utilizados nos histogramas utilizamos o script “best_profits.sh” que nos vai permitir retirar dos ficheiros gerados com o ficheiro “job_selection.c” a informação acerca do maior lucro total obtidos, quando o lucro é ignorado.



Histograma 1 : Lucros para número mecanográfico 98498



Histograma 2 : Lucros para número mecanográfico 98512



Histograma 3: Lucros para número mecanográfico 98513

Com o código Matlab implementado “best_profits.m” geramos, então, os histogramas visualizados, Histograma 1, 2 e 3, que nos mostram o número de vezes que cada lucro apareceu (em intervalos). Estes foram gerados pela função Matlab *histfit*, que constrói um histograma e gera uma linha vermelha que representa a distribuição normal.

Visto que, ao utilizar esta função, foram gerados histogramas mais a linha que representa a distribuição normal, podemos concluir que, ao mostrarmos o número de ocorrências que cada intervalo de lucros aparece, estes não seguem uma distribuição normal. Para seguirem uma distribuição normal, então teriam de ter forma parecida a uma gaussiana e teriam de se encontram todos os intervalos com valores acima da linha vermelha, o que não se verifica.

Para o número mecanográfico 98498, podemos concluir que os melhores lucros totais gerados têm um número de ocorrências superior nos intervalos de 20000 e 31000, aparecendo o mesmo número de vezes, sendo que para intervalos um pouco inferiores a 20000 e no intervalo entre 21000 até 30000 se gera um números de ocorrências um pouco inferior ao número de ocorrências máximo.

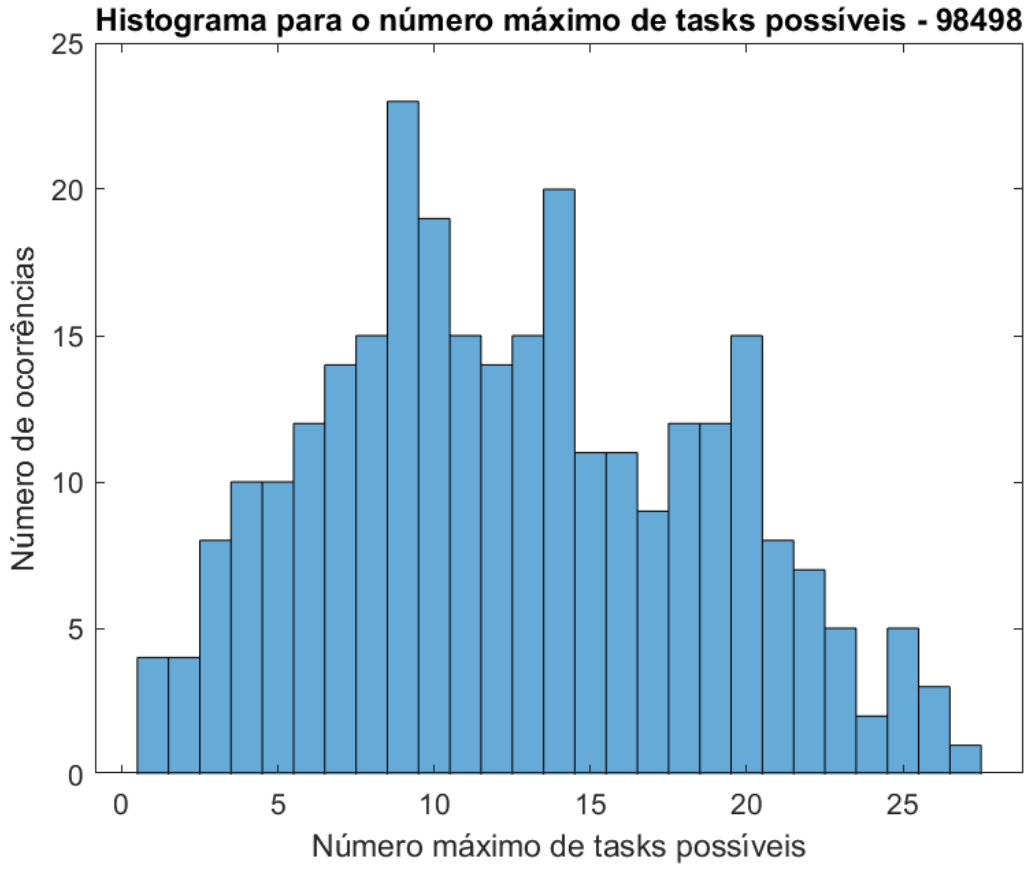
Para o número mecanográfico 98512, observa-se que o maior número de ocorrências que se gera é no intervalo um pouco inferior a 30000, estando o intervalo um pouco inferior a 20000, também, muito próximo do número máximo de ocorrências gerado.

Para o número mecanográfico 98513, observamos que o maior número de ocorrências que foi gerado se encontra no intervalo de 20000, contudo podemos observar que no intervalo para valores um pouco abaixo de 30000, o número de ocorrências gerados foi próximo do valor máximo de ocorrências gerado.

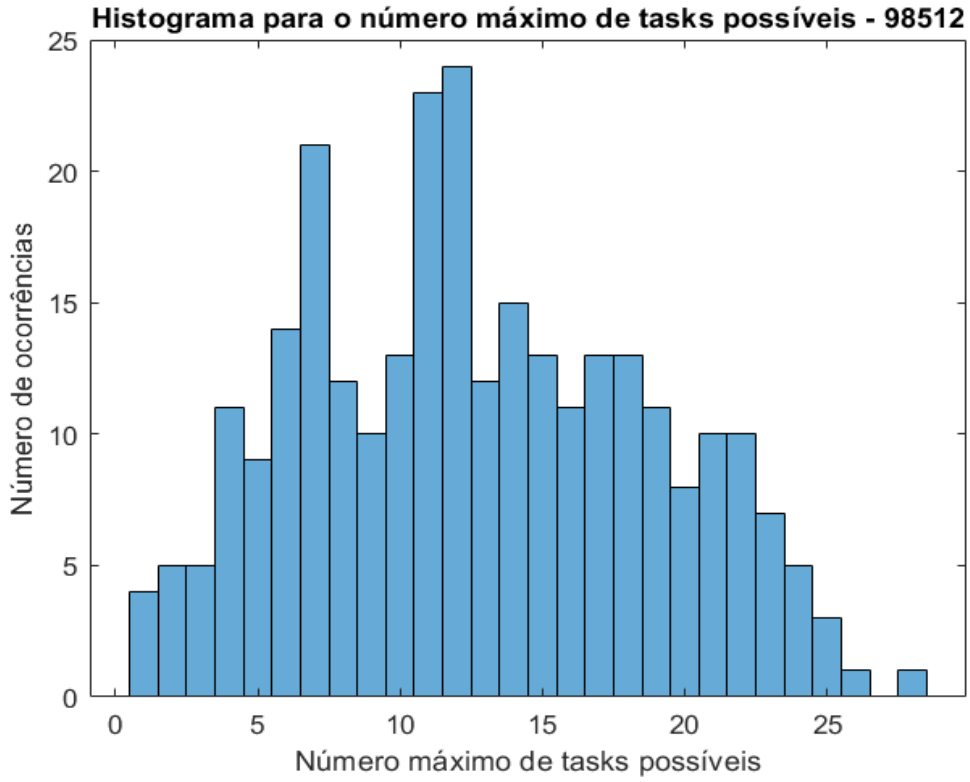
3.6 – Histogramas número máximo de tarefas

Para os ficheiros criados com o script “job_selection_do_all.sh” (Pág. 28 Fig. 6), ao utilizarmos para a variável I o valor 1, vamos obter ficheiros com informações diferentes das informações obtidas para I=0, como por exemplo, o máximo de tarefas possíveis e soluções possíveis.

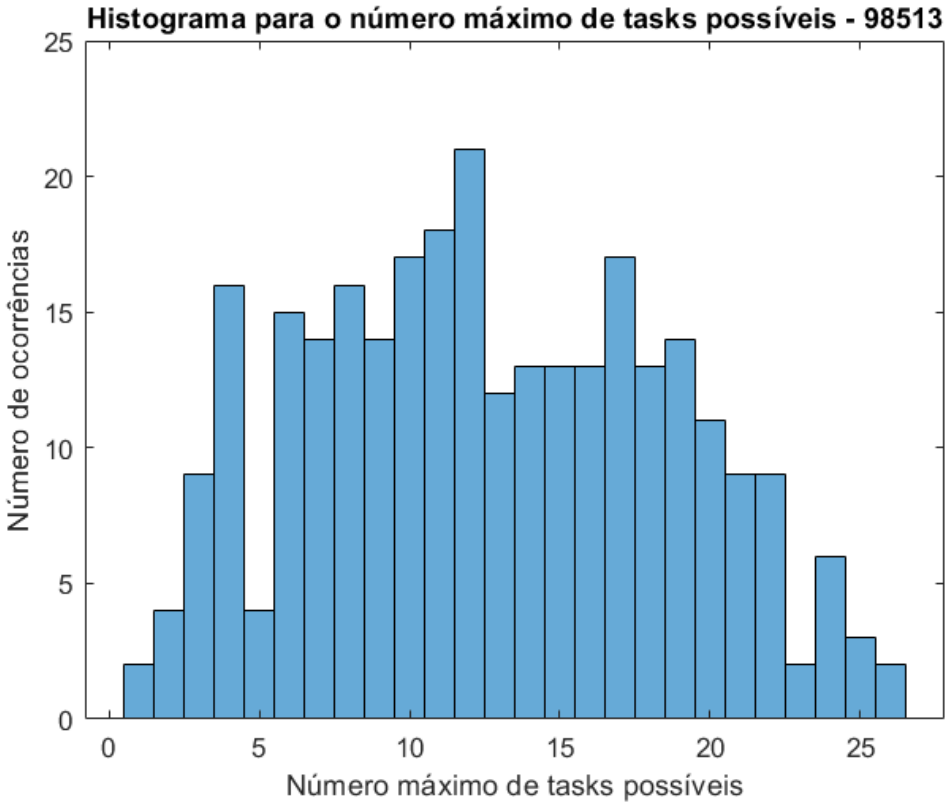
Para criarmos os histogramas do número máximo de tarefas possíveis, utilizamos o ficheiro “maxTasks.m” (Pág 32 Fig. 18/ Pág 33 Figs. 19 e 20), implementado em Matlab e para obter os valores que foram utilizados nos histogramas utilizamos o script “maxTasks_1.sh” (Pág 32 Fig. 17) que nos vai permitir retirar dos ficheiros gerados com o ficheiro “job_selection.c” a informação acerca do número máximo de tarefas que é possível realizar, quando o lucro é ignorado.



Histograma 4 : Número máximo de tarefas possíveis para número mecanográfico 98498



Histograma 5 : Número máximo de tarefas possíveis para número mecanográfico 98512



Histograma 6 : Número máximo de tarefas possíveis para número mecanográfico 98513

Com o código Matlab implementado “maxTasks_1.m” geramos, então, os histogramas visualizados, Histograma 4, 5 e 6, que nos mostram o número de vezes que o número máximo de tarefas apareceu.

Observamos, então, que para o número mecanográfico 98498, o número máximo de tarefas possíveis que apareceu mais vezes foi o número 9, aparecendo também muitas vezes o valor 10 e 14.

Para o número mecanográfico 98512, o número máximo de tarefas possíveis que apareceu mais vezes foi o número 12, aparecendo também muitas vezes o valor 7 e 11.

Para o número mecanográfico 98513, o número máximo de tarefas possíveis que apareceu mais vezes foi o número 12, verificando-se para este número mecanográfico, uma maior equivalência entre valores, visto que os próximos valores mais gerados se encontram com um número de ocorrências muito próximos uns dos outros.

3.7 – Gráficos das soluções e número máximo de tarefas

De modo, a visualizarmos as soluções possíveis para cada tarefa em cada número mecanográfico, geramos um ficheiro “solutions_1.sh” (Pág 34 Fig. 21) que nos permite retirar, quando o lucro é ignorado, as informações acerca das soluções de cada tarefa realizada.

Para se poder comparar as soluções geradas com o número máximo de tarefas possíveis utilizamos o ficheiro “maxTasks_1.sh” (Pág 32 Fig.17) onde fizemos um gráfico, para cada número mecanográfico.

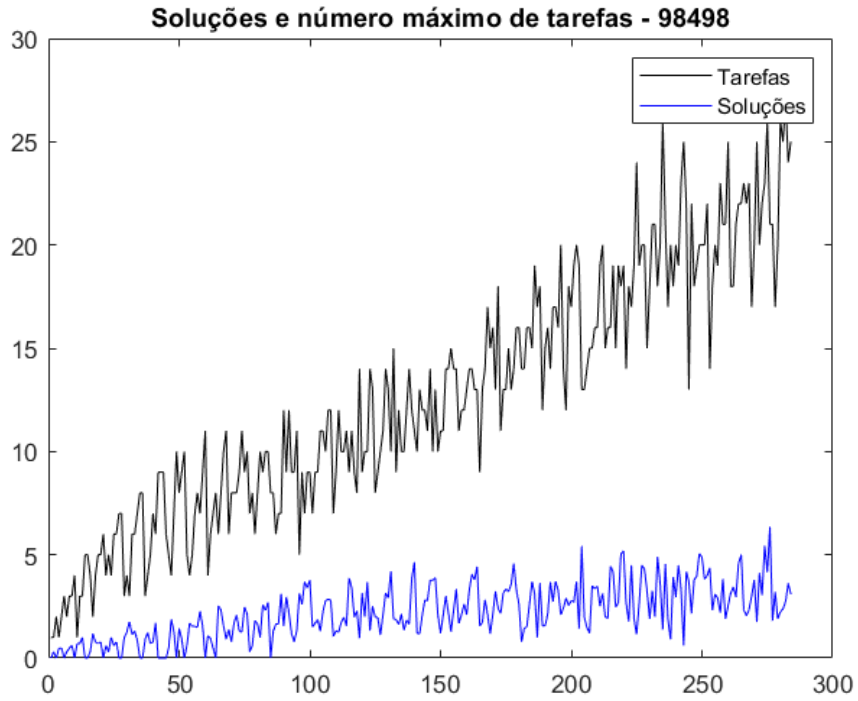


Gráfico 4: Soluções e número máximo de tarefas para o número mecanográfico 98498

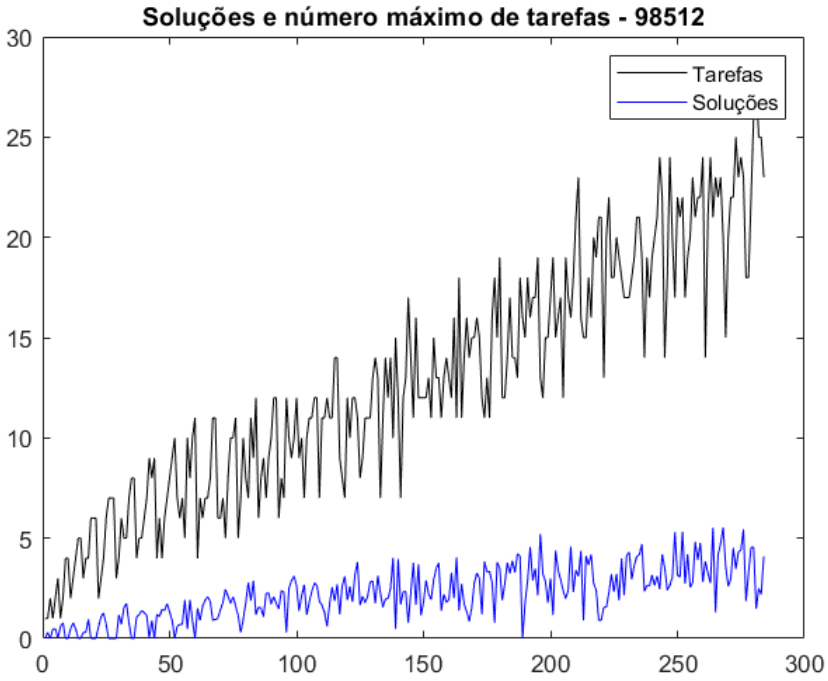


Gráfico 5: Soluções e número máximo de tarefas para o número mecanográfico 98512

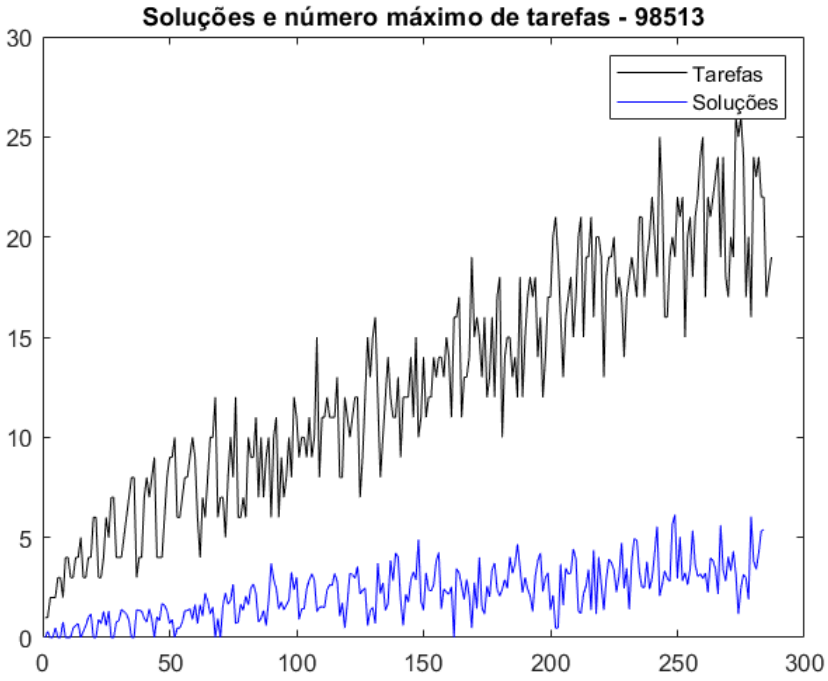


Gráfico 6: Soluções e número máximo de tarefas para o número mecanográfico 98513

Como as soluções, a partir de certo ponto, começam a crescer de um modo muito superior ao número máximo de tarefas possíveis, para que fosse possível observar estas duas informações, decidimos colocar no *plot* das soluções, log10(solutions), para que fosse possível visualizar tudo no mesmo gráfico.

Concluindo-se assim, que tanto para as soluções como para o número máximo de tarefas possíveis, os valores vão aumentando gradualmente à medida que o número de tarefas vai aumentando.

4. Apêndice

```bash
#! /bin/bash
NMec=98498
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..39}; do
  for P in {1..8}; do
    if (( $T >= $P )); then
      f=$(printf "%06d/%02d_%02d_%d.txt" $NMec $T $P $I)
      if [ ! -e $f ]; then
        echo $f
        ./prog $NMec $T $P $I
      fi
    fi
  done
done
```

Figura 6: Código bash do job_selection_do_all.sh.

```bash
#! /bin/bash
cd ./098498
#Retirar de cada ficheiro o lucro
#sed serve para ir apenas aos ficheiros que terminam apenas em _0.txt, sendo estes aqueles que tem os profits
grep "Best total profit =" *.txt | sed -e 's/_0.txt:Best total profit =//' -e 's/_/ /' > best_profits.txt
```

Figura 7: Código bash do best_profits.sh.

```
clear all
close all

%Meter path onde se tem o ficheiro best_profit guardado
%Cada variavel fica com 3 colunas
prof98498 = load('098498\best_profits.txt' , 'r'); % Tasks  Programadores Lucro
prof98512 = load('098512\best_profits.txt' , 'r');
prof98513 = load('098513\best_profits.txt' , 'r');
```

Figura 8: Parte do código Matlab do best_profits.m.

```
%% Histogramas

figure(1)
% Ir à terceira coluna de cada variável para retirar apenas os lucros
profit498 = prof98498(: , 3);
% Histograma com distribuição normal
histfit(profit498);
title('Lucros para nº Mecanográfico - 98498');
xlabel('Lucro');
ylabel('Nº de Ocorrências');

figure(2)
profit512 = prof98512(: , 3);
histfit(profit512);
title('Lucros para nº Mecanográfico - 98512');
xlabel('Lucro');
ylabel('Nº de Ocorrências');

figure(3)
profit513 = prof98513(: , 3);
histfit(profit513);
title('Lucros para nº Mecanográfico - 98513');
xlabel('Lucro');
ylabel('Nº de Ocorrências');
```

Figura 9: Parte do código Matlab do best_profits.m.

```
%% Código usado para guardar informação útil para a construção de tabelas dos bests profits em word
tabela_98498 = cell(40,9);
tabela_98498{1,1} = ' Tasks / Programadores ';  %cada linha representa as tasks
                                                %cada coluna representa os programadores

for linha = 2:40    %preenchimento da legenda de cada linha
    tabela_98498{linha,1} = linha - 1;
end

for coluna = 2:9    %preenchimento da legenda de cada coluna
    tabela_98498{1,coluna} = coluna - 1;
end

tabela_98513 = tabela_98498; %copiar os cabeçalhos
tabela_98512 = tabela_98498; %copiar os cabeçalhos

%Preencher cada elemento, com o valor de best profit das respetivas tasks e
%programadores, de acordo com os cabeçalhos adicionados:

for line = 1:284
    tabela_98498{prof98498(line , 1) + 1,prof98498(line , 2) + 1} = prof98498(line , 3);
end

for line = 1:284
    tabela_98513{prof98513(line , 1) + 1,prof98513(line , 2) + 1} = prof98513(line , 3);
end

for line = 1:284
    tabela_98512{prof98512(line , 1) + 1 ,prof98512(line , 2) + 1} = prof98512(line , 3);
end
```

Figura 10: Parte do código Matlab do best_profits.m.

```
%% Gráficos dos best profits

figure
plot3(prof98498(:,1),prof98498(:,2),profit498,'go-');
hold on
plot3(prof98513(:,1),prof98513(:,2),profit513,'r*-');
plot3(prof98512(:,1),prof98512(:,2),profit512,'bs-');
hold off
title('Best profits')
xlabel('Tarefas');
ylabel('Programadores');
zlabel('Best profit');
legend('98498','98513', '98512')
```

Figura 11: Parte do código Matlab do best_profits.m.

```
#! /bin/bash
cd 098498

grep "Solution time =" *.txt | sed -e 's/_0.txt:Solution time =//' -e 's/_/ /' >t1_98498

mv t1_98498 ..
```

Figura 12: Código bash de extract_data.sh.

```
clear all
close all

fid = fopen('t1_98513','r');
cell_98513 = textscan(fid,'%s %s','delimiter',newline); %cell_98513{1,1} = informação dos tempos quando I = 0
                                                        %cell_98513{1,2} = informação dos tempos quando I = 1
fclose(fid);

fid = fopen('t1_98498','r');
cell_98498 = textscan(fid,'%s %s','delimiter',newline); %cell_98498{1,1} = informação dos tempos quando I = 0
                                                        %cell_98498{1,2} = informação dos tempos quando I = 1
fclose(fid);

fid = fopen('t1_98512','r');
cell_98512 = textscan(fid,'%s %s','delimiter',newline); %cell_98512{1,1} = informação dos tempos quando I = 0
                                                        %cell_98512{1,2} = informação dos tempos quando I = 1
fclose(fid);
```

Figura 13: Parte do Código Matlab de graficos_tempos.m.

```
%% Gráfico dos tempos de execução para I=0

n = cellfun(@str2double, (cellfun(@(x) x(1:2),cell_98513{1,1},'UniformOutput',false)) ); %n = tasks de 98513
p = cellfun(@str2double, (cellfun(@(x) x(4:5),cell_98513{1,1},'UniformOutput',false)) ); %p = programadores de 98513
t = cellfun(@str2double, (cellfun(@(x) x(7:end),cell_98513{1,1},'UniformOutput',false)) ); %t = tempos de 98513
figure
plot3(n,p,log10(t),'r*-');

hold on
n = cellfun(@str2double, (cellfun(@(x) x(1:2),cell_98498{1,1},'UniformOutput',false)) ); %n = tasks de 98498
p = cellfun(@str2double, (cellfun(@(x) x(4:5),cell_98498{1,1},'UniformOutput',false)) ); %p = programadores de 98498
t = cellfun(@str2double, (cellfun(@(x) x(7:end),cell_98498{1,1},'UniformOutput',false)) ); %t = tempos de 98498
plot3(n,p,log10(t),'go-');

n = cellfun(@str2double, (cellfun(@(x) x(1:2),cell_98512{1,1},'UniformOutput',false)) ); %n = tasks de 98512
p = cellfun(@str2double, (cellfun(@(x) x(4:5),cell_98512{1,1},'UniformOutput',false)) ); %p = programadores de 98512
t = cellfun(@str2double, (cellfun(@(x) x(7:end),cell_98512{1,1},'UniformOutput',false)) ); %t = tempos de 98512
plot3(n,p,log10(t),'bs-');

hold off
title('Tempos de execuçao, I=0')
xlabel('Tarefas');
ylabel('Programadores');
zlabel('Tempo (s)');
legend('98513', '98498', '98512')
```

Figura 14: Parte do Código Matlab de graficos_tempos.m.

```
%% Gráfico dos tempos de execução para I=1

n = cellfun(@str2double, (cellfun(@(x) x(1:2),cell_98513{1,2},'UniformOutput',false)) ); %n = tasks de 98513
p = cellfun(@str2double, (cellfun(@(x) x(4:5),cell_98513{1,2},'UniformOutput',false)) ); %p = programadores de 98513
t = cellfun(@str2double, (cellfun(@(x) x(29:end),cell_98513{1,2},'UniformOutput',false)) );  %t = tempos de 98513
figure
plot3(n,p,log10(t),'r*-');

hold on
n = cellfun(@str2double, (cellfun(@(x) x(1:2),cell_98498{1,2},'UniformOutput',false)) ); %n = tasks de 98498
p = cellfun(@str2double, (cellfun(@(x) x(4:5),cell_98498{1,2},'UniformOutput',false)) ); %p = programadores de 98498
t = cellfun(@str2double, (cellfun(@(x) x(29:end),cell_98498{1,2},'UniformOutput',false)) ); %t = tempos de 98498
plot3(n,p,log10(t),'go-');

n = cellfun(@str2double, (cellfun(@(x) x(1:2),cell_98512{1,2},'UniformOutput',false)) );  %n = tasks de 98512
p = cellfun(@str2double, (cellfun(@(x) x(4:5),cell_98512{1,2},'UniformOutput',false)) ); %p = programadores de 98512
t = cellfun(@str2double, (cellfun(@(x) x(29:end),cell_98512{1,2},'UniformOutput',false)) ); %t = tempos de 98512
plot3(n,p,log10(t),'bs-');

hold off
title('Tempos de execuçao, I=1')
xlabel('Tarefas');
ylabel('Programadores');
zlabel('Tempo (s)');
legend('98513', '98498', '98512')
```

Figura 15: Parte do Código Matlab de graficos_tempos.m.

```bash
#! /bin/bash
cd ./098513
#Retirar de cada ficheiro o num_viables, o número de subconjuntos de tarefas válidas
#somar todos os num_viables, de modo a obter o numero de subconjuntos de tarefas válidas para cada numero mecanográfico
total_viable_tasks=$( grep "Viable job selections =" *.txt | awk '{print $5}' | paste -sd+ - | bc)

echo $total_viable_tasks
```

Figura 16: Código Bash do total_valid_tasks.bash.

```bash
#! /bin/bash
cd ./098498

#Retirar de cada ficheiro o número máximo de tasks
#sed serve para ir apenas aos ficheiros que terminam apenas em _1.txt, sendo estes aqueles que tem o número máximo de tasks
grep "Maximum possible tasks =" *.txt | sed -e 's/_1.txt:Maximum possible tasks =//' -e 's/_/ /' > maxTasks.txt
```

Figura 17: Código Bash do maxTasks_1.sh.

```
%% maxTasks.m
clear all
close all

%Meter path onde se tem o ficheiro maxTasks guardado
task98498 = load('098498\maxTasks.txt' , 'r');
task98512 = load('098512\maxTasks.txt' , 'r');
task98513 = load('098513\maxTasks.txt' , 'r');
sol98498 = load('098498\solutions.txt' , 'r');
sol98512 = load('098512\solutions.txt' , 'r');
sol98513 = load('098513\solutions.txt' , 'r');

maxTask498 = task98498(:,3);
figure(1)
histogram(maxTask498)
title('Histograma para o número máximo de tasks possíveis - 98498')
xlabel('Número máximo de tasks possíveis')
ylabel('Número de ocorrências')

maxTask512 = task98512(:,3);
figure(2)
histogram(maxTask512)
title('Histograma para o número máximo de tasks possíveis - 98512')
xlabel('Número máximo de tasks possíveis')
ylabel('Número de ocorrências')
```

Figura 18: Parte código Matlab de maxTasks.m

```
maxTask513 = task98513(:,3);
figure(3)
histogram(maxTask513)
title('Histograma para o número máximo de tasks possíveis - 98513')
xlabel('Número máximo de tasks possíveis')
ylabel('Número de ocorrências')


soluct498 = sol98498(: , 3);
soluct512 = sol98512(: , 3);
soluct513 = sol98513(: , 3);

figure(4)
plot(maxTask498 , "k-");
hold on;
plot(log10(soluct498), "b-");
legend("Tarefas" , "Soluções");
hold off;
title("Soluções e número máximo de tarefas - 98498")
```

Figura 19: Parte código Matlab de maxTasks.m

```
figure(5)
plot(maxTask512 , "k-");
hold on;
plot(log10(soluct512), "b-");
legend("Tarefas" , "Soluções");
hold off;
title("Soluções e número máximo de tarefas - 98512")

figure(6)
plot(maxTask513 , "k-");
hold on;
plot(soluct513, "b-");
legend("Tarefas" , "Soluções");
hold off;
title("Soluções e número máximo de tarefas - 98513")
```

Figura 20: Parte final código Matlab de maxTasks.m

```
#! /bin/bash
cd ./098513

#Retirar de cada ficheiro o número máximo de tasks
#sed serve para ir apenas aos ficheiros que terminam apenas em _1.txt, sendo estes aqueles que tem o número máximo de tasks
grep "Solutions =" *.txt | sed -e 's/_1.txt:Solutions =//' -e 's/_/ /' > solutions.txt
```

Figura 21: Código Bash do Solutions_1.sh

5. Conclusão

O ficheiro “job_selection.c” permite-nos, com base no número mecanográfico inserido, gerar diferentes lucros para cada tarefa, sendo que o problema baseia-se na obtenção do subconjunto com o melhor lucro total.

Para além das opções obrigatórias, o nosso grupo fez também algumas variantes opcionais, tais como, a construção de uma função recursiva “best_subset”, visto que concluímos que a nossa primeira implementação (*Brute Force*), embora funcional, demora muito mais tempo do que com a função recursiva, aspeto que tentamos evitar visto que, um dos objetivos do problema era obter um bom tempo de execução. Fizemos um script “maxTasks_1.sh” que nos gera um ficheiro com o maior número de tarefas possíveis para cada combinação de tarefas e programadores e fizemos, ainda, um contador que gera o número de tarefas válidas para cada número mecanográfico e histogramas para quando os lucros são ignorados.

Concluímos então, que quando o lucro não é ignorado, geralmente, quanto maior é o número de tarefas e de programadores, o maior lucro obtido vai aumentando gradualmente, havendo algumas exceções onde o lucro diminui. Isto porque, quando há mais programadores poderão ser realizadas mais tarefas, por conseguinte, o lucro total será maior.

Quando o lucro é ignorado, observamos que, tanto o número de soluções como o número máximo de tarefas possíveis aumenta gradualmente à medida que o número de tarefas aumenta.

Neste ponto da realização do relatório podemos afirmar que conseguimos alcançar todos os objetivos que o problema propunha, sendo desta forma muito satisfatória a realização deste problema.