CalcSM: Uma calculadora simplificada para programação de máquinas simples

Projeto da disciplina de Programação de Produção Intermitente

5 de setembro de 2011

Ricardo Ryoiti Sugawara Júnior

1. Introdução

O CalcSM consiste num programa que efetua cálculos simultâneos de diversas funções objetivo relativas à programação de ordens em máquinas simples. A partir de um arquivo de dados, são informados os parâmetros de uma ou mais instâncias de programação, contendo os parâmetros de cada uma das ordens que as compõem. É então escolhido um entre seis métodos de ordenação e busca que determinam a sequência a ser efetivamente calculada. Os resultados são apresentados em modo tabulado e existe a opção de gerar um gráfico de Gantt com a sequência final. Detalhes dos cálculos podem ser apresentados com a opção de saída detalhada.

Este documento descreve de modo sumário as funcionalidades do programa e o meio de utilização do mesmo, incluindo exemplos de execução e explicações simplificadas sobre as estruturas de dados, e técnicas empregadas nas buscas, ordenações e cálculos.

2. Estruturas de dados básicas

O programa foi escrito na linguagem C para ser executado em linha de comando, o que facilita o reuso do mesmo em *scripts* ou por chamadas desde um programa terceiro. Todos os cálculos são tratados em torno de duas estruturas de dados básicas: `instancias` e `ordens`. Essas estruturas abrigam os parâmetros informados e calculados das instâncias e ordens, respectivamente, e são apresentados a seguir.

Existem, nas duas estruturas, ponteiros de memória que fazem o vínculo entre as mesmas. Ambas são tratadas como listas duplamente ligadas.

Estrutura `instancias`:

```
typedef struct instancias {
    int inst;    /** Número da instância. */
    int nordens; /** Número de ordens nesta instância. */

    int Cmax;  /** Makespan.
                * The makespan, defined as max(C1 , . . . , Cn ), is
                * equivalent to the completion time of the last job to leave
                * the system. A minimum makespan usually implies a good
                * utilization of the machine(s).
                */

    int sum_C;  /** Flow time. */
    int sum_wC; /** Total weighted completion time.
                 * The sum of the weighted completion times of the n jobs gives
                 * an indication of the total holding or inventory costs
                 * incurred by the schedule. The sum of the completion times is
                 * in the literature often referred to as the flow time. The
                 * total weighted completion time is then referred to as the
                 * weighted flow time.*/
    int sum_wCmR; /** Discounted flow time.
```

```
                    * Definida como SUM w*(C_j - R_j) onde R é o release */
    int sum_U; /** Total number of tardy jobs. */
    int sum_wU;/** Weighted number of tardy jobs. */
    int sum_T; /** Total tardiness. */
    int sum_wT;/** Total weighted tardiness.
                     * This is also a more general cost function than the total
                     * weighted completion time. */
    int Tmax;  /** Maximum tardiness. */
    int Lmax;  /** Maximum lateness.
                * The maximum lateness, Lmax , is defined as
                * max(L1 , . . . , Ln ). It measures the worst violation
                * of the due dates. */
    int sum_wTpwE; /** Total weighted earliness and tardiness. */

    /* objetivos abaixo não foram originalmente solicitados. */
    int sum_E; /** total earliness */
    int sum_wE;/** total weighted earliness */
    int Emax;  /** Maximum earliness. */

    struct ordens *ohead;   /** link com instancia */

    struct instancias *next;    /** proxima instancia */
    struct instancias *prev;    /** instancia anterior */

} instancias;
```

Estrutura `ordens`:

```
    typedef struct ordens {

        int inst;      /** instancia */
        int ordem;     /** ordem */
        int r;         /** release date */
        int d;         /** due date */
        int w;         /** weight */
        int wT;        /** weighted tardiness */
        int wE;        /** weighted earliness */
        int p;         /** processing time */
        float z;       /** dispatch rule ranking */

        /* parametros do roteiro */
        int seq;       /** sequencia */
        int startt;    /** start time */
        int realloc;   /** realocada por restricao? */

        /** link com as instancias */
        struct instancias *instptr;

        /* DL */
        struct ordens *next;    /** proxima ordem */
        struct ordens *prev;    /** ordem anterior */
    } ordens;
```

3. Comandos e funcionalidades

A execução do programa ocorre com uma única chamada ao executável `calcsm`. Com o comando `-h` são listados todos os parâmetros e opções aceitas e a sintaxe de emprego dos mesmos. Também são apresentados os algoritmos de busca/sequenciamento disponíveis e as funções objetivas calculadas ou minimizadas. A saída a seguir apresenta o resultado da tela de auxílio do programa.

```
    calcsm 0.1: calcula sequenciamento de máquina única

    Uso: calcsm [-h] [-v] [-f ARQUIVO] [-o ARQUIVO] [-i SEQ] -s [METODO]
```

```
  -h              imprime esta ajuda e sai

  -v              configura saída detalhada
  -f              configura entrada (padrão: stdin)
  -p              plotar gráfico de gantt
  -i SEQ          indica número da sequência (padrão: todas)
  -s METODO       metodo de ordenação (padrão: seq)

  -o OBJ          função objetivo (padrão: atraso ponderado)

  -c NUM          número de sorteios (método rand. padrão: sorteio simples)

  -r              Reordenar seqüência se inviável (padrão: atrasar ordens)

Métodos de sequenciamento disponíveis:
  rand            randômico
  seq             sequência imposta (coluna 9 da entrada)
  disp            regra de liberação W(LBE+LBT)
  annl            simmulated annealing
  edd             earliest due date first
  enum            enumeration - exhaustive search
  twkr            regra de liberação TWKR-BY-TIS

Funções objetivo disponíveis:
  1        Makespan (Cmax)
  2        Tempo de fluxo (sum_C)
  3        Fluxo ponderado (sum_wC)
  4        Fluxo desc. p. (sum_wCmR)
  5        Ord. atrasadas (sum_U)
  6        Ord. atras. pond (sum_wU)
  7        Atraso total (sum_T)
  8        Atras. pond. tot (sum_wT)
  9        Atraso maximo (Tmax)
  10       Lateness maximo (Lmax)
  11       Atras+adiant pond (sum_wTpwE)
  12       Adiant. total (sum_E)
  13       Adiant. ponderado (sum_wE)
  14       Adiant. maximo (Emax)
```

Os seguintes detalhamentos descrevem a função dos parâmetros relevantes.

`-v` A saída do programa exibirá, conforme os cálculos forem sendo efetuados, o resultado parcial durante as iterações. Ao sortear sequências aleatoriamente, por exemplo, são exibidas cada uma das sequências e o resultado da função objetiva, indicando se é ou não uma melhor solução até a presente iteração. Em caso de regras de liberação (*dispatch rules*, opções `disp` e `twkr` para o parâmetro `-s`), é apresentado o cálculo dos *rankings* e as decisões sobre alocação das ordens.

`-f` Indica qual o arquivo de dados contém os dados das instâncias e das ordens (ver tópico seguinte). Em caso de omissão, o programa aceita a entrada de modo interativo (entrada via teclado ou redirecionamento). Isto permite que o software seja chamado e alimentado por um programa terceiro, inserindo os dados de modo dinâmico.

`-p` Monta um gráfico de Gantt com as sequências finais que geraram os resultados (ver tópico Gráfico de Gantt).

`-i SEQ` Indica qual das instâncias `SEQ` da entrada de dados serão consideradas. Se omitida, todas as identificadas nos dados de entrada serão calculadas

simultaneamente. Se informada uma instância inexistente, o resultado do programa será vazio.

`-s METODO` Escolhe a técnica para alocação de ordens. Estão disponíveis os métodos listados na saída do auxílio do programa.

`-o OBJ` Escolhe a função objetiva a ser minimizada. Apesar de todas as funções objetidas listadas no auxílio do programa serem calculadas simultaneamente, apenas uma (a indicada no parâmetro) é considerada nos métodos de busca para minimização (opções `rand`, `annl`, e `enum` do parâmetro `-s`).

`-c NUM` Este parâmetro limita o número de iterações/sorteios dos métodos de sorteio randômico (`-s rand`) e busca exaustiva por enumeração completa (`-s enum`).

`-r` Caso o sorteio ou a sequência imposta não seja viável em função de uma ordem não estar disponível (*release time*) no momento de sua alocação, este parâmetro instrui o programa a alocar a próxima ordem, colocando a atual (não disponível) numa lista de espera para alocação assim que possível. Isto efetivamente altera a sequência final gerando uma sequência sempre viável, porém distinta. Por isto, não é o comportamento padrão, que consiste em atrasar a ordem até que a mesma fique disponível (introduzindo ociosidade).

4. Formato dos dados de entrada

Os dados de entrada devem ser tabulados, contendo os seguintes parâmetros em sequência:

a. Instância
b. Ordem
c. Release time
d. Due date
e. Weight
f. Tardiness weight
g. Earliness weight
h. Processing time
i. Sequência imposta

O exemplo abaixo ilustra as primeiras linhas de um arquivo de dados compatível. É apresentada uma instância de número `1`, com ordens numeradas de `1` a `4`. Todas as linhas que se iniciam com `#` são consideradas comentários aos dados e ignoradas.

```
#Inst   Ord     Rj      Dj      Wj      WTj     WEj     Pj      Seq
1       1       0       20      1       1       1       5       2
1       2       0       5       1       1       1       4       1
1       3       0       7       1       1       1       3       4
1       4       0       19      1       1       1       10      3
```

A última coluna (9, sequência imposta) deve conter a sequência em que as ordens são agendadas forçadamente por meio do método `-s seq`. No exemplo, a sequência imposta é `2, 1, 4, 3`.

5. Exemplos de utilização e informações sobre os resultados

A seguir são apresentados exemplos de utilização do programa e a respectiva saída de dados. As informações da saída serão comentadas.

a. Sequência imposta

Para a sequência imposta, deve-se escolher o parâmetro `-s seq`. No exemplo a seguir, será também escolhida a instância de número 1, com a sequência imposta `1, 2, 3, 4, 5, 6` (sequência é mesma ordenação do número das ordens).

Logo após a coluna `seq.`, que indica qual a sequência final da ordem, pode haver um dos seguintes caracteres (marcados em azul no exemplo abaixo):

`E` Indica *earliness*. A ordem foi adiantada em relação ao *due date*.

`T` Indica *tardiness*. Ordem atrasada.

`R` Ordem realocada na sequência (ver parâmetro `-r`).

Os demais dados da ordem são apresentados na lista que resume a sequência final. Em seguida é apresentado o resumo da instância, com seu número de ordens e funções objetivo calculados.

```
$ ./calcsm -f dados.txt -i 1 -s seq
calcsm 0.1: calcula sequenciamento de máquina única

                R E S U M O   D A   S E Q U Ê N C I A
                   start   release    due     proc    job weights
   inst   ord  seq  time    time       date    time    w_j w_T w_E
      1     1    1E 0       0          20      5         1   1   1
      1     2    2T 5       0          5       4         1   1   1
      1     3    3T 9       0          7       3         1   1   1
      1     4    4T 12      0          19      10        1   1   1
      1     5    5T 22      0          12      2         1   1   1
      1     6    6T 24      0          21      11        1   1   1

                R E S U M O   D A   I N S T Â N C I A

           Instancia |    1
      Total de ordens |    6
---------------------------
 1          Makespan |   35
 2    Tempo de fluxo |  107
 3   Fluxo ponderado |  107
 4    Fluxo desc. p. |  107
 5    Ord. atrasadas |    5
 6  Ord. atras. pond |    5
 7      Atraso total |   38
 8  Atras. pond. tot |   38
 9     Atraso maximo |   14
10   Lateness maximo |   14
11 Atras+adiant pond |   53
12     Adiant. total |   15
13 Adiant. ponderado |   15
14    Adiant. Maximo |   15
```

Se não for explicitada a instância, todos os dados obtidos do arquivo são processados. O exemplo abaixo ilustra o resultado, com a listagem das ordens parcialmente suprimida para brevidade.

```
$ ./calcsm -f dados.txt -s seq
calcsm 0.1: calcula sequenciamento de máquina única

                R E S U M O   D A   S E Q U Ê N C I A

                   start   release    due     proc    job weights
   inst   ord  seq  time    time       date    time    w_j w_T w_E
      1     1    1E 0       0          20      5         1   1   1
      1     2    2T 5       0          5       4         1   1   1
    (suprimido...)
     12     3    3T 20      0          1       13        1   1   1
     12     4    4T 33      0          12      4         1  12   1

                R E S U M O   D A   I N S T Â N C I A

           Instancia |    1|    2|    3|    4|    5|    6|    7|    8|    9|   10|   11|   12
     Total de ordens |    6|    4|    4|    5|    3|    5|    4|    5|    4|    6|    7|    4
-----------------------------------------------------------------------------------------------
 1          Makespan |   35|   17|   22|   31|  110|  560|   10| 5600|   44|   77|   89|   37
 2    Tempo de fluxo |  107|   39|   62|   97|  141| 1658|   20|16580|  111|  246|  330|  100
 3   Fluxo ponderado |  107|   39|   62|   97|10102| 1658|   20|16580|  111|  246|  330|  100
 4    Fluxo desc. p. |  107|   39|   53|   97|10102| 1658|   20|16580|  111|  246|  330|  100
 5    Ord. atrasadas |    5|    2|    2|    3|    1|    3|    2|    3|    2|    2|    2|    4
 6  Ord. atras. pond |    5|    2|    2|    3|    1|    3|    2|    3|    8|    6|    9|   39
 7      Atraso total |   38|    8|   21|   17|   10|  398|    5| 3980|   27|   19|   43|   81
 8  Atras. pond. tot |   38|    8|   21|   17|   10|  398|    5| 3980|  115|   58|  193|  632
 9     Atraso maximo |   14|    7|   14|   10|   10|  223|    4| 2230|   17|   10|   22|   32
10   Lateness maximo |   14|    7|   14|   10|   10|  223|    4| 2230|   17|   10|   22|   32
11 Atras+adiant pond |   53|   18|   21|   21|  179|  603|   10| 4766|  125|  148|  335|  632
12     Adiant. total |   15|   10|    0|    4|  169|  205|    5|  786|   10|   90|   74|    0
13 Adiant. ponderado |   15|   10|    0|    4|  169|  205|    5|  786|   10|   90|  142|    0
14    Adiant. Maximo |   15|    6|    0|    2|   89|  139|    4|  786|    6|   46|   34|    0
```

b. Busca randômica

A busca randômica deve receber o parâmetro `-c` para indicar quantos sorteios serão efetuados. Caso contrário, apenas uma única sequência é gerada. No exemplo a seguir, é também colocado o parâmetro `-v` para detalhar a saída. Observe a identificação de melhoria nos objetivos e o resultado final após as `20` iterações solicitadas. Por brevidade, os dados das ordens e instâncias foram suprimidos.

```
$ ./calcsm -f dados.txt -i 1 -s rand -c 20 -v
calcsm 0.1: calcula sequenciamento de máquina única

Iniciando com método: randômico (rand)...

BUSCA RANDÔMICA: Instância 1. Função objetivo: sum_wT (Atras. pond. tot), até 20
iterações.

Sorteio    1. Inst:   1 Obj:    48. Seq:  1  2  3  6  4  5 (melhor objetivo)
Sorteio    2. Inst:   1 Obj:    57. Seq:  3  1  4  6  5  2
Sorteio    3. Inst:   1 Obj:    57. Seq:  3  1  4  6  5  2
Sorteio    4. Inst:   1 Obj:    49. Seq:  4  3  5  1  6  2
Sorteio    5. Inst:   1 Obj:    49. Seq:  4  3  2  5  6  1
Sorteio    6. Inst:   1 Obj:    78. Seq:  6  1  4  5  2  3
Sorteio    7. Inst:   1 Obj:    58. Seq:  2  4  6  3  5  1
Sorteio    8. Inst:   1 Obj:    43. Seq:  3  4  2  5  6  1 (melhor objetivo)
Sorteio    9. Inst:   1 Obj:    57. Seq:  2  4  6  5  3  1
Sorteio   10. Inst:   1 Obj:    70. Seq:  6  2  4  3  5  1
Sorteio   11. Inst:   1 Obj:    37. Seq:  1  2  5  4  3  6 (melhor objetivo)
Sorteio   12. Inst:   1 Obj:    53. Seq:  4  5  1  3  6  2
Sorteio   13. Inst:   1 Obj:    66. Seq:  5  6  4  1  3  2
Sorteio   14. Inst:   1 Obj:    37. Seq:  3  4  2  5  1  6
```

```
    Sorteio   15. Inst:   1 Obj:    58. Seq:  6  5  3  4  1  2
    Sorteio   16. Inst:   1 Obj:    29. Seq:  3  1  2  5  6  4 (melhor objetivo)
    Sorteio   17. Inst:   1 Obj:    74. Seq:  1  6  2  4  5  3
    Sorteio   18. Inst:   1 Obj:    78. Seq:  1  4  6  2  5  3
    Sorteio   19. Inst:   1 Obj:    44. Seq:  2  1  3  6  4  5
    Sorteio   20. Inst:   1 Obj:    63. Seq:  6  2  5  4  1  3

    Melhor solução objetivo: 29. Seq: 3 1 2 5 6 4

    (resto da saída suprimida)
```

c. Busca exaustiva

A busca exaustiva pode ser limitada com o parâmetro `-c`, de modo semelhante à busca aleatória. A seguir é apresentado, novamente com o detalhamento `-v`, a saída parcial para uma execução de busca cuja limitação de iterações é maior que o número de combinações possíveis (enumeração completa).

```
    $ ./calcsm -f dados.txt -i 1 -s enum -c 1000 -v
    calcsm 0.1: calcula sequenciamento de máquina única

    Iniciando com método: enumeration - exhaustive search (enum)...

    BUSCA EXAUSTIVA: Instância 1. Função objetivo: sum_wT (Atras. pond. tot), até 1000
iterações.

    Sequência    1. Inst:   1 Obj:    38. Seq:  1  2  3  4  5  6 (melhor objetivo)
    Sequência    2. Inst:   1 Obj:    47. Seq:  1  2  3  4  6  5
    Sequência    3. Inst:   1 Obj:    30. Seq:  1  2  3  5  4  6 (melhor objetivo)
    Sequência    4. Inst:   1 Obj:    48. Seq:  1  2  3  6  4  5
    (suprimido...)
    Sequência  717. Inst:   1 Obj:    61. Seq:  5  4  6  3  2  1
    Sequência  718. Inst:   1 Obj:    72. Seq:  6  4  5  3  2  1
    Sequência  719. Inst:   1 Obj:    63. Seq:  5  6  4  3  2  1
    Sequência  720. Inst:   1 Obj:    64. Seq:  6  5  4  3  2  1

    Melhor solução objetivo: 18. Seq: 2 3 5 4 1 6

    (resto da saída suprimida)
```

d. Simulated Annealing

A implementação de simmulated annealing utiliza os seguintes parâmetros:

```
    #define N_TRIES 10          /* how many points do we try before stepping */
    #define ITERS_FIXED_T 1     /* how many iterations for each T? */
    #define STEP_SIZE 1.0       /* max step size in random walk */
    #define K 1.0               /* Boltzmann constant */
    #define T_INITIAL 5000.0    /* initial temperature */
    #define MU_T 1.002          /* damping factor for temperature */
    #define T_MIN 5.0e-1        /* stop temperature */
```

É utilizada a bibiloteca de nome “GNU Scientific Library” [1]. A função de caminho à vizinhança consiste num passo aleatório a partir da sequência existente. O passo inicial é tomado como a sequência imposta no arquivo de dados (coluna 9). No método `-s annl` não é considerado o limite de iterações no parâmetro `-c`, pois o controle é efetuado por meio da temperatura.

Um exemplo de saída segue:

```
    $ ./calcsm -f dados.txt -i 1 -s annl
```

```
    calcsm 0.1: calcula sequenciamento de máquina única

    Iniciando com método: simmulated annealing (annl)...

    SIMULATED ANNEALING: Instância 1. Função objetivo: sum_wT (Atras. pond. tot), até 1
iterações.

    #-iter  #-evals   temperature     position   energy
        0         2          5000  [ Seq:  1  3  2  4  5  6 ]            37            37
        1         3       4990.02  [ Seq:  3  1  2  4  5  6 ]            29            29
        2         4       4980.06  [ Seq:  3  1  2  6  5  4 ]            31            29
        3         5       4970.12  [ Seq:  1  3  2  6  5  4 ]            39            29
        4         6        4960.2  [ Seq:  4  3  2  6  5  1 ]            52            29
        5         7        4950.3  [ Seq:  4  1  2  6  5  3 ]            32            29
        6         8       4940.42  [ Seq:  5  1  2  6  4  3 ]            29            29
        7         9       4930.56  [ Seq:  2  1  5  6  4  3 ]            44            29
        8        10       4920.72  [ Seq:  2  1  3  6  4  5 ]            27            27

    (resto da saída suprimida)
```

e. Regras de liberação (*dispatch rules*)

São implementadas três regras de liberação como métodos de sequenciamento, a saber:

`-s lbelbt` **W(LBE+LBT)**: Weighted Lower Bound Earliness + Lower Bound Tardiness

Este método é descrito na referência [2] e foi escolhido como o método para minimizar a função objetivo 11 (atraso e adiantamento ponderado). Consiste em calcular um parâmetro de nome

$LBCT_j$: Lower Bound Completion Time for job *j*.

que determina o limite inferior para o custo de atraso e adiantamento num instante *T*. Ele é definido como:

$$LBCT_j = T + ECT_j + LST_j$$

Como no nosso caso há apenas uma tarefa por job, existem simplificações codificadas no algoritmo. Efetivamente,

$$ECT_j = T - p_j$$

e

$$LST_j = T$$

de modo que

$$LBCT_j = T + p_j$$

Com isto, é calculada a disjunção

$$LBSC_j = max\{ (LBCT_j - d_j; 0) \} w_{Tj} - max \{(d_j - LBCT_j; 0 \} w_{Ej}$$

sendo agendado primeiro o job com maior $LBSC_j$.

O algoritmo também é mencionado na dissertação [3].

`-s edd` **EDD**: Earliest Due Date first

Este é um simples algoritmo que procura reduzir a probabilidade de atraso ao alocar ordens com menor data de entrega primeiro. Está incluído no programa para enriquecer eventuais estudos comparativos.

`-s twkr` **TWKR-BY-TIS**: Total Work Content Remaining by Time in System first

Este algoritmo também foi testado no artigo [2] e é objeto também da dissertação [3]. Foi considerado para o tratamento do atraso e adiantamento ponderado. Também recebe simplificações em função do job conter apenas uma tarefa, porém demonstrou ser menos eficiente no conjunto de dados testado. Mesmo assim, foi mantido para permitir comparações e estudos com o programa desenvolvido.

O exemplo abaixo ilustra o trabalho do algoritmo W(LBE+LBT) no programa:

```
$ ./calcsm -f dados.txt -i 2 -s disp -v
calcsm 0.1: calcula sequenciamento de máquina única

Iniciando com método: regra de liberação W(LBE+LBT) (disp)...

W(LBE+LBT): Sequência 1 Ordem 1 LBSC = 4, d = 8, wE = 1, wT = 1, d_j = -4.00.
W(LBE+LBT): Sequência 1 Ordem 2 LBSC = 2, d = 12, wE = 1, wT = 1, d_j = -10.00.
W(LBE+LBT): Sequência 1 Ordem 3 LBSC = 6, d = 11, wE = 1, wT = 1, d_j = -5.00.
W(LBE+LBT): Sequência 1 Ordem 4 LBSC = 5, d = 10, wE = 1, wT = 1, d_j = -5.00.
W(LBE+LBT): Alocando ordem 1 na Sequência 1.
W(LBE+LBT): Sequência 2 Ordem 2 LBSC = 2, d = 12, wE = 1, wT = 1, d_j = -10.00.
W(LBE+LBT): Sequência 2 Ordem 3 LBSC = 6, d = 11, wE = 1, wT = 1, d_j = -5.00.
W(LBE+LBT): Sequência 2 Ordem 4 LBSC = 5, d = 10, wE = 1, wT = 1, d_j = -5.00.
W(LBE+LBT): Alocando ordem 3 na Sequência 2.
W(LBE+LBT): Sequência 3 Ordem 2 LBSC = 2, d = 12, wE = 1, wT = 1, d_j = -10.00.
W(LBE+LBT): Sequência 3 Ordem 4 LBSC = 5, d = 10, wE = 1, wT = 1, d_j = -5.00.
W(LBE+LBT): Alocando ordem 4 na Sequência 3.
W(LBE+LBT): Sequência 4 Ordem 2 LBSC = 2, d = 12, wE = 1, wT = 1, d_j = -10.00.
W(LBE+LBT): Alocando ordem 2 na Sequência 4.
Fim.
(resto da saída suprimida...)
```

Para efeitos comparativos, segue um exemplo de confronto entre os algoritmos `enum`, `disp` e `twkr` para o objetivo `11`.

Instancia	1	2	3	4	5	6	7	8	9	10	11	12
DISP: 11 Atras+adiant pond	34	15	18	23	21	572	7	5916	140	67	188	652
TWKR: 11 Atras+adiant pond	49	19	18	34	179	754	10	7484	137	111	425	480
ENUM: 11 Atras+adiant pond	22	12	17	27	179	628	7	5016	77	80	71	408

6. Gráfico de Gantt

O programa está preparado para gerar um gráfico de Gantt com as sequências finais. Para tanto, basta indicar o parâmetro -p que, ao final dos cálculos, é aberta uma janela com o gráfico resultante. A geração é efetuada com o programa GNUplot [5] com o auxílio de um *script* [6]. A seguir é apresentado um exemplo para algumas instâncias escolhidas.

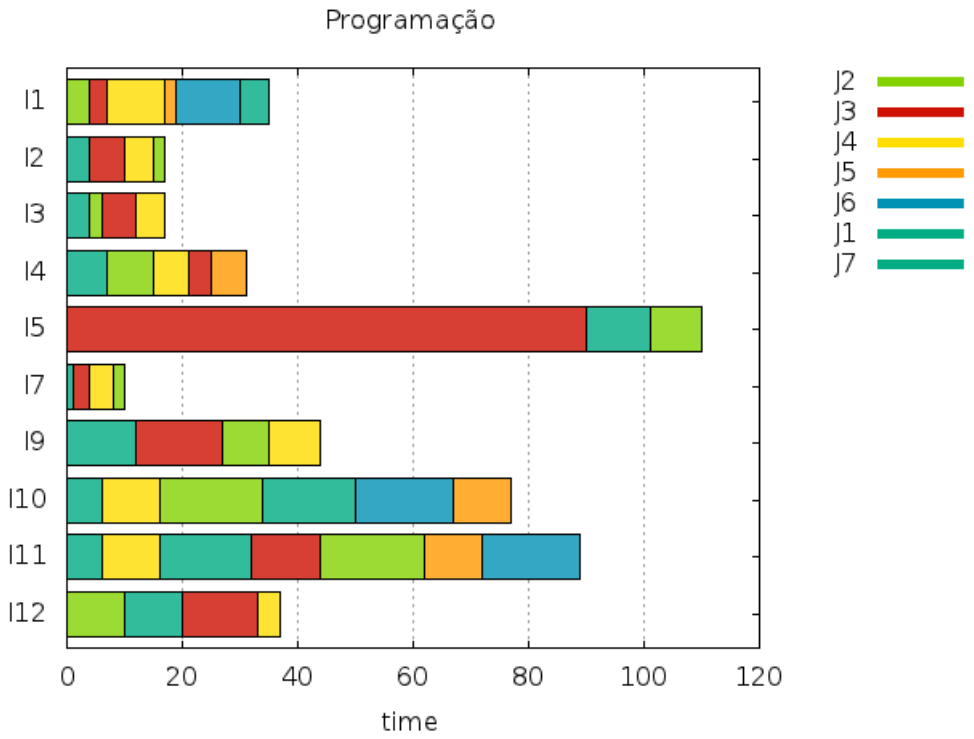

Figura 1: Gráfico de Gantt resultante dos cálculos das instâncias (1-6, 7, 9-12). Método `annl`.

7. Bibliografia

[1] The GNU Scientific Library (GSL). http://www.gnu.org/s/gsl Acesso em 29/08/2011.

[2] S. Thiagarajan, Chandrasekharan Rajendran, *Scheduling in dynamic assembly job-shops to minimize the sum of weighted earliness, weighted tardiness and weighted flowtime of jobs*, Computers & Industrial Engineering 49 (2005) 463–503.

[3] M. T. Pereira, *Proposta de um Modelo de Simulação Computacional para a Programação de Operações em Sistemas Assembly Shop*, (2009).

[4] *IBM WebSphere MQ Software*, http://www-01.ibm.com/software/integration/wmq/. Acesso em 29/03/2011.

[5] GNUplot. http://www.gnuplot.info Acesso em 29/08/2011.

[6] A Python script for drawing Gantt charts with Gnuplot. http://se.wtb.tue.nl/sewiki/wonham/gantt.py Acesso em 29/08/2011.