Otimização do Processo de Contratação de Funcionários e Gestão Horária utilizando PLR

João Maia and Miguel Sandim

FEUP-PLOG, Turma 3MIEIC2, Grupo Contrata_Hor_Func_2
Faculdade de Engenharia da Universidade do Porto,
Rua Dr. Roberto Frias, Porto, Portugal

Resumo A temática deste artigo debruça-se sobre a otimização do processo de contratação de funcionários e gestão dos seus horários no contexto de uma empresa. Considera-se que neste processo uma empresa apresenta uma janela temporal diária de trabalho com necessidades mínimas de número de funcionários que podem variar durante o dia e restrições quanto à sua atividade. A solução ótima neste contexto garante que o custo suportado (total de salários a pagar aos trabalhadores) é mínimo, requerindo pelo menos as necessidades mínimas de funcionários. O presente artigo propõe um algoritmo de obtenção da solução ótima usando PLR (Programação Lógica com Restrições) em SICSTUS PROLOG.

Keywords: otimização linear, programação em lógica com restrições, gestão de funcionários, gestão de horários

1 Introdução

No âmbito da Unidade Curricular de Programação em Lógica foi proposta a resolução de um problema de otimização, fazendo uso de Programação Lógica com Restrições em PROLOG. O problema abordado consiste na otimização do processo de contratação de funcionários e gestão dos seus horários de trabalho de acordo com os recursos necessários de contratação e restrições impostos.

No decorrer deste artigo será analisada a abordagem da solução em PLR (nomeadamente variáveis de decisão e restrições associadas) e explicada a visualização da solução ótima. Serão também analisadas medições experimentais do tempo de execução do algoritmo com a variação de certos parâmetros e feito um parecer sobre a sua fiabilidade.

2 Descrição do Problema

O problema de otimização em análise prende-se com a rentabilização do processo de gestão da contratação de funcionários a tempo inteiro/parcial numa empresa, podendo ser divido nos seguintes sub-problemas:

1. Gestão do processo de recrutamento de funcionários a tempo inteiro/parcial, segundo as restrições de contratação definidas.
2. Gestão do horário de trabalho de cada funcionário face às necessidades da empresa e às limitações impostas.

2.1 Descrição das Variáveis do Problema

Considera-se que uma empresa apresenta uma janela temporal de trabalho diário, representada pelo intervalo $[\alpha, \omega]$, sendo α a primeira hora de trabalho do dia e ω a última hora de trabalho. Para cada empresa existe uma função $f(t) : t \in [\alpha, \omega]$ que modela as necessidades mínimas de funcionários requeridas em função de t. Dada a natureza diária do problema, conclui-se que t é um valor discreto, e que portanto $t \in [0, 23] \wedge t \in \mathbb{N}_0$.

Tabela 1. Exemplo de uma função $f(t)$ para uma determinada empresa, com $\alpha = 9$ e $\omega = 16$.

t	9	10	11	12	13	14	15	16
f(t)	4	3	3	6	5	6	8	8

Existem dois tipos de funcionários para efeito de preenchimento das necessidades laborais:

- **Funcionários contratados a tempo inteiro**, i.e. que trabalham durante toda a janela temporal definida. Este tipo de funcionários tem direito a uma hora de almoço no intervalo [$startLunch$, $endLunch$], ganhando $FullSal$ unidades/hora (inclusive à hora de almoço). Contudo podem abdicar da sua hora de almoço, ganhando um bónus de $FullBonus$ unidades.
- **Funcionários contratados a tempo parcial**, i.e. que trabalham durante $PartialHours$ horas durante a janela temporal definida, ganhando $PartialSal$ unidades/hora. Esta hierarquia de funcionários pode trabalhar $PartialHours$ $+1$ horas a troco de mais $PartialBonus$ unidades.

Contudo, impede-se que a soma do número de funcionários sem hora de almoço e com hora extra ultrapasse $MaxExtra$. Restringe-se também o número de funcionários em tempo parcial: este não deverá ser superior a $MaxPartial$ nem maior que $MaxPartialRatio$ % do número de funcionários a tempo inteiro.

2.2 Solução do Problema

Considera-se como solução do problema: uma função $g(t) : t \in [\alpha, \omega]$ que modela o número efetivo de funcionários em cada hora da janela temporal de trabalho da empresa e informação relativa ao tipo de cada funcionário contratado, assim como o seu horário de trabalho e salário.

Solução Ótima A solução ótima do problema corresponde a um configuração que garanta um custo mínimo à empresa, i.e. o total de salários pagos aos funcionários deverá ser mínimo.

Assim, define-se como função objetivo:

$$min\left(\sum_{i=1}^{N} S_i\right) \tag{1}$$

sendo N o número total de funcionários e S_i o salário do funcionário i.

3 Abordagem em PLR

A proposta de resolução deste problema em PLR apresentada nas secções seguintes utiliza CLP(FD) [1] em SICSTUS PROLOG.

3.1 Variáveis de Decisão

Funcionários a Tempo Inteiro

- $FullLun_i$ – Indica a hora de almoço de um funcionário i, podendo ser -1 se o funcionário trabalha na hora de almoço ou -2 se este não existe (descrita no código fonte com o nome $FullWorkersLunchHours$).
  Domínio: $\{-1\} \cup \{-2\} \cup [startLunch, endLunch]$

Funcionários a Tempo Parcial

- $PartStart_i$ – Indica a hora de início do horário de um funcionário i, sendo -1 no caso em que o funcionário não existe (descrita no código fonte com o nome $PartialWorkersStartHour$).
  Domínio: $\{-1\} \cup [\alpha, \omega]$.

- $PartExtra_i$ – Indica o número de horas extra do horário do funcionário i (descrita no código fonte com o nome $PartialWorkersExtraHour$).
  Domínio: $\{0\} \cup \{1\}$.

Detalhes de Implementação Dado o número de variáveis de decisão depender do número de funcionários contratados e este não ser conhecido *à priori*, na implementação em PROLOG o tamanho da lista *FullWorkersLunchHours* foi definido como $2 \cdot max(f(t))$ e o tamanho das listas *PartialWorkersStartHour* e *PartialWorkersExtraHour* como $minimum(MaxPartial,\ 2 \cdot max(f(t)))$.

3.2 Restrições Aplicadas

Restrições sobre Funcionários a Tempo Inteiro

$$\begin{aligned}&(FullLun_i = -2 \wedge FullSal_i = 0)\\ &\quad \vee (FullLun_i = -1 \wedge FullSal_i = SalaryBase)\\ &\quad \vee (FullLun_i >= 0 \wedge FullSal_i = SalaryPlus), \forall i\end{aligned} \tag{2}$$

$$FullLun_i \leq FullLun_{i+1}, \forall i \tag{3}$$

A restrição (2) garante a correta correlação entre características do funcionário e o seu salário, sendo que $FullSal_i$ é uma variável de domínio auxiliar representando o salário do funcionário i, $SalaryBase$ é o valor do salário para um funcionário com hora de almoço e $SalaryPlus$ é o valor para um funcionário sem hora de almoço. A restrição (3) garante que não são consideradas no espaço de procura potenciais soluções simétricas.

Estas restrições encontram-se implementadas no predicado *applyFullRestrictions/6*.

Restrições sobre Funcionários a Tempo Parcial

$$\begin{aligned}&(PartStart_i = -1 \wedge PartExtra_i = 0 \wedge PartSal_i = 0)\\ &\quad \vee (PartStart_i > -1 \wedge PartExtra_i = 0 \wedge PartSal_i = SalaryBase)\\ &\quad \vee (PartStart_i > -1 \wedge PartExtra_i = 1 \wedge PartSal_i = SalaryPlus), \forall i\end{aligned} \tag{4}$$

$$PartStart_i \leq PartStart_{i+1}, \forall i \tag{5}$$

A restrição (4) garante a correta correlação entre as características do funcionário e o seu salário, sendo que $PartSal_i$ é uma variável de domínio auxiliar representando o salário do funcionário i, $SalaryBase$ é o valor do salário para um funcionário sem hora extra e $SalaryPlus$ é o valor para um funcionário com hora extra. A restrição (5) garante que não são consideradas no espaço de procura potenciais soluções simétricas.

Estas restrições encontram-se implementadas no predicado *applyPartialRestrictions/5*.

Restrições sobre Funcionários

$$FullNoLun + \sum_{j=1}^{N} PartExtra_j \leq MaxExtra \tag{6}$$

$$PartNum \leq \frac{MaxPartialRatio}{100} FullNum \tag{7}$$

$$g(t) = Fullin_t + Partin_t, \forall t \in [\alpha, \omega] \tag{8}$$

$$f(t) \leq g(t), \forall t \in [\alpha, \omega] \tag{9}$$

A restrição (6) garante que a soma dos funcionários que trabalham na hora de almoço e dos funcionários que têm hora extra seja menor que $MaxExtra$ especificado na definição do problema, sendo $FullNoLun$ o total de elementos $FullLun_i : FullLun_i = -1$. A restrição (7) garante que o número efetivo de funcionários a tempo parcial é menor do que $MaxPartialRatio$ % do número de funcionários a tempo inteiro, sendo $PartNum$ o total de elementos $PartStart_i : PartStart_i \neq -1$ e $FullNum$ o total de elementos $FullLun_i : FullLun_i \neq -2$. A restrição (8) garante que o $g(t)$ toma o valor da soma do número de funcionários a tempo inteiro e parcial a trabalhar à hora t, representado simbolicamente por $Fullin_t$ e $Partin_t$. A restrição (9) garante que a solução apresenta pelo menos as necessidades mínimas da empresa em toda a sua janela temporal diária.

Estas restrições encontram-se implementadas no predicado *applyCommonRestrictions/8*.

3.3 Função de Avaliação

Tal como definido em (1), a função objetivo procura minimizar o custo associado a cada solução (total de salários a pagar aos funcionários).

Este custo é calculado segundo o seguinte predicado:

Código fonte 1.1. Definição do predicado de cálculo de custo de uma solução.

```
calculateCost(FullWorkersSalary, PartialWorkersSalary, TotalSalary) :-
    sum(FullWorkersSalary, #=, TotalFullWorkersSalary),
    sum(PartialWorkersSalary, #=, TotalPartialWorkersSalary),
    TotalSalary #= TotalFullWorkersSalary + TotalPartialWorkersSalary.
```

3.4 Estratégia de Pesquisa

Na etiquetagem das variáveis de decisão foram utilizadas as seguintes *flags* do predicado *labeling/2*:

- *ffc*: as *flags ff* e *ffc* revelaram ser as mais otimizadas no processo de procura quando comparadas com a *leftmost* e *max*. Isto poderá ser explicado pelo facto de permitirem a escolha no processo de *labeling* das variáveis com menor domínio, o que terá mais probabilidade de diminuir os domínios das restantes variáveis. Sendo que ambas conduziram a bons resultados experimentais, selecionou-se a *flag ffc*.
- *step*: conclui-se que a exploração do domínio de cada variável não será mais eficiente utilizando heurísticas específicadas (com as *flags bisect* ou *enum*).
- *up*: a exploração ascendente do domínio das variáveis de decisião permitiu uma execução experimental mais rápida do que uma exploração descendente, tendo-se optado pela *flag up*.
- *minimize(TotalSalary)*: como a etiquetagem tem como único objetivo a procura pela solução ótima, foi utilizada a opção *minimize* que acelera a pesquisa e origina apenas uma solução final.

4 Visualização da Solução

Na figura 1 é apresentada a visualização da solução ótima em SICSTUS PROLOG para a instância do problema fornecida no enunciado, em que: $StartLunch = 12$, $EndLunch = 13$, $FullSal = 80$, $FullBonus = 40$, $PartialHours = 3$, $PartialSal = 50$, $PartialBonus = 60$, $MaxExtra = 2$, $MaxPartial = 5$, $MaxPartialRatio = 30$.

Depois de calculada a solução ótima do problema, é mostrado o tempo de execução do programa, recorrendo ao predicado *showStatistics/0*. É visualizada de seguida a solução encontrada com o predicado *printSolution/9*, que por sua vez utiliza os seguintes predicados:

- *printFullWorkers/4* e *printPartialWorkers/4*: visualizam informação relativa ao número de funcionários a tempo inteiro/parcial contratados e ainda o número de funcionários em cada situação possível.
- *printTimetable/3*: visualiza a função $g(t)$ obtida.
- *printScheduleBoard/8*: visualiza uma tabela com a totalidade dos horários calculados acompanhados pelo salário do funcionário.

```
*** Sicstus: Statistics ***
Time passed (ms): 172


*** Solution ***

* Textual display of the schedule for each type of worker:
7 full workers
-> 2 full workers do extra hour at lunch and receive a salary of 680
-> 2 full workers lunch at 12 and receive a salary of 640
-> 3 full workers lunch at 13 and receive a salary of 640

2 partial workers
-> A partial worker starts at 11, ends at 14 and receives a salary of 150
-> A partial worker starts at 14, ends at 17 and receives a salary of 150


* Final timetable:
Hours      9    10   11   12   13   14   15   16
Workers    7    7    8    6    5    8    8    8


* Graphical solution of all the workers' schedule:
     9    10   11   12   13   14   15   16   Salary
     X    X    X         X    X    X    X    640
     X    X    X         X    X    X    X    640
     X    X    X    X         X    X    X    640
     X    X    X    X         X    X    X    640
     X    X    X    X         X    X    X    640
     X    X    X    X    X    X    X    X    680
     X    X    X    X    X    X    X    X    680
               X    X    X                   150
                              X    X    X    150

Total Salary Cost: 4860
```

Figura 1. Visualização da solução ótima da instância do problema do enunciado no SICSTUS.

5 Resultados

Apresenta-se de seguida a análise da eficiência temporal da solução em PLR através de medições experimentais variando certos parâmetros do problema [1].

5.1 Número de funcionários requeridos

Para testar a influência do aumento do número de funcionários necessários pela empresa em cada hora (contradomínio de $f(t)$) na execução do algoritmo, foram feitas medições do tempo de execução do algoritmo variando x, sendo $f(t) = x, \forall t \in [\alpha, \omega]$.

Nas medições efetuadas foi considerado $MaxPartial = 5$, pelo que durante os testes existe apenas variação do tamanho da lista de variáveis de decisão relativa aos funcionários a tempo inteiro: *FullWorkersLunchHours*. Os dados experimentais confirmam o resultado teórico, dado que é esperado que exista uma relação polinomial entre x e o tempo de execução devido ao aumento do número de variáveis de decisão.

[1] Medições realizadas num portátil com um CPU Intel I7 a 1.6Ghz com 4 *cores* e 4GB de RAM.

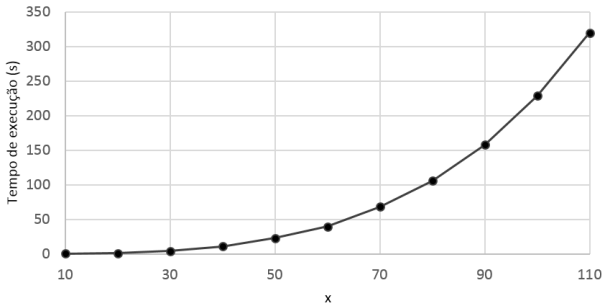

Figura 2. Gráfico do tempo de execução do algoritmo (em segundos) em função de x.

5.2 Numero máximo de funcionários a tempo parcial

Com o objetivo de estudar a influência do número máximo de funcionários a tempo parcial ($MaxPartial$), foram feitas medições do tempo de execução do programa em duas situações:

- Quando $f(t) = 10, \forall t \in [\alpha, \omega]$ (gráfico da esquerda da figura 3).
- Quando $f(t) = 15, \forall t \in [\alpha, \omega]$ (gráfico da direita da figura 3).

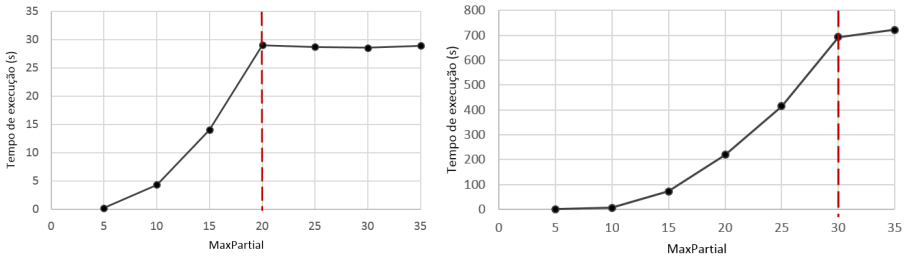

Figura 3. Gráfico do tempo de execução (em segundos) do algoritmo em função de $MaxPartial$.

Durante as medições foi variado o parâmetro $MaxPartial$ que influencia diretamente o tamanho das listas de variáveis de decisão $PartialWorkersStartHour$ e $PartialWorkersExtraHour$, relativas aos funcionários a tempo parcial.

Analisando os gráficos da figura 3, o tempo de execução mostra-se bastante sensível à variação de $MaxPartial$ especialmente quando em conjugação com a variação do número de funcionários requeridos, podendo uma diferença de apenas cinco funcionários provocar alterações bastante acentuadas no tempo de procura da solução. Esta diferença pode ser explicada pelo facto destes dois tipos de variação aumentarem o tamanho das três listas de variáveis de decisão, o que contribui para um aumento bastante significativo do tempo de procura.

É possível também notar-se que o tempo de execução permanece constante após um certo valor de $MaxPartial$ (marcado com uma linha vermelha nos gráficos). Este comportamento advém da definição do tamanho das listas de variáveis de decisão dos funcionários a tempo parcial, tal como explicado na

secção 3.1. Nesse caso, para valores de $MaxPartial$ à esquerda da linha o tamanho das listas aumenta, e para valores à direita da linha o tamanho das listas mantêm-se constante por ser ter atingido $2 \cdot max(f(t))$.

6 Conclusões e Trabalho Futuro

O resultados experimentais permitem concluir que a proposta de resolução obtida cumpre com todos os requisitos do problema e garante para qualquer instância do problema uma única solução ótima.

Contudo, acredita-se que o algoritmo desenvolvido não é capaz de lidar com "problemas da vida real", já que uma empresa de tamanho considerável poderá ter de gerir horários de centenas de funcionários e apresentar grandes necessidades de funcionários por hora, com um máximo de funcionários a tempo parcial que poderá ser igualmente alto. O algoritmo proposto porém, conseguiu resolver a instância do problema proposta no enunciado em cerca de $141ms$ e permite gestão de horários em tempo aceitável quando as necessidades dos funcionários se inserem na ordem das dezenas. A natureza do problema também dificultou a implementação de uma proposta algorítmica totalmente eficaz.

Uma possível ordem de trabalhos futuros neste algoritmo poderiam incluir: melhor estratégia de definição do maior número possível de funcionários (crucial para instanciação das listas de variáveis de decisão, dado que o número de funcionários a contratar não é conhecido à *priori*) e desenvolvimento de uma estratégia alternativa à definição de variáveis de decisão por cada possível funcionário (que acaba por ser algo "custosa").

Referências

1. Carlsson M., Ottosson G., Carlson B. *An Open-Ended Finite Domain Constraint Solver* Proc. Programming Languages: Implementations, Logics, and Programs, 1997.

A Medições Experimentais

Tabela 2. Medições experimentais relativas ao gráfico da figura 2.

x	**Tempo de execução (s)**
10	0.187
20	1.031
30	3.969
40	10.469
50	22.453
60	39.407
70	68.125
80	105.735
90	157.484
100	228.36
110	319.67

Tabela 3. Medições experimentais relativas ao gráfico da esquerda da figura 3.

MaxPartial	**Tempo de execução (s)**
5	0.204
10	4.297
15	14.016
20	29.016
25	28.672
30	28.547
35	28.891

Tabela 4. Medições experimentais relativas ao gráfico da direita da figura 3.

MaxPartial	**Tempo de execução (s)**
5	0.391
10	6.109
15	71.234
20	219.282
25	414.171
30	693.719
35	721.828

B Código Fonte

B.1 Ficheiros de teste

Para permitir a execução do algoritmo com diferentes instâncias do problema, foi adotado um sistema de ficheiros de factos (correspondentes a parâmetros do problema).

O ficheiro *probEx.pl* contém os parâmetros do problema *default* fornecido no enunciado.

Código fonte 1.2. code/probEx.pl

```
% Slots input:
input_slots([4, 3, 3, 6, 5, 6, 8, 8]).
input_startWork(9).
input_endWork(16).

input_maxExtraWorkers(2).

% *** Full time workers: ***

input_fullSalaryPerHour(80).
input_fullBonus(40).

% Lunch hour:
input_lunchHourList([12,13]).

% *** Partial time workers: ***
input_partialWorkHours(3).
input_partialMaxWorkers(5).
input_partialWorkersRatio(30). %

input_partialSalaryPerHour(50).
input_partialBonus(60).
```

Código fonte 1.3. code/probMedium.pl

```
% Slots input:
input_slots([10, 10, 10, 10, 10, 10, 10, 10]).
input_startWork(9).
input_endWork(16).

input_maxExtraWorkers(30).

% *** Full time workers: ***

input_fullSalaryPerHour(80).
input_fullBonus(40).

% Lunch hour:
input_lunchHourList([12,13]).

% *** Partial time workers: ***
input_partialWorkHours(3).
input_partialMaxWorkers(10).
input_partialWorkersRatio(30). %

input_partialSalaryPerHour(50).
input_partialBonus(60).
```

Código fonte 1.4. code/probBig.pl

```
% Slots input:
input_slots([40, 50, 40, 40, 50, 50, 30, 20, 5]).
input_startWork(9).
input_endWork(17).

input_maxExtraWorkers(2).

% *** Full time workers: ***

input_fullSalaryPerHour(80).
input_fullBonus(40).

% Lunch hour:
input_lunchHourList([12,13]).

% *** Partial time workers: ***
```

```
input_partialWorkHours(3).
input_partialMaxWorkers(6).
input_partialWorkersRatio(30). %

input_partialSalaryPerHour(50).
input_partialBonus(60).
```

Código fonte 1.5. code/probBig2.pl

```
% Slots input:
input_slots([40, 32, 60, 79, 35, 46, 18, 100, 20, 2]).
input_startWork(9).
input_endWork(18).

input_maxExtraWorkers(2).

% *** Full time workers: ***

input_fullSalaryPerHour(80).
input_fullBonus(40).

% Lunch hour:
input_lunchHourList([12,13]).

% *** Partial time workers: ***
input_partialWorkHours(3).
input_partialMaxWorkers(6).
input_partialWorkersRatio(30). %

input_partialSalaryPerHour(50).
input_partialBonus(60).
```

B.2 Solução em PLR

Código fonte 1.6. code/hor.pl

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).

:- consult(probEx).

% ************************************************************
% ************************ Input *****************************
% ************************************************************
totalWorkHours(NumberHours) :-
        input_slots(SlotsInput),
        length(SlotsInput, NumberHours).

% ********************** Full Workers ************************

fullSalaryBase(Salary) :-
        input_fullSalaryPerHour(PerHour),
        totalWorkHours(Hours),
        Salary is PerHour * Hours.

fullSalaryPlus(Salary) :-
        fullSalaryBase(SalaryBase),
        input_fullBonus(Bonus),
        Salary is SalaryBase + Bonus.

startLunchHour(Start) :-
        input_lunchHourList(Slots),
        nth1(1, Slots, Start).

```

```
endLunchHour(End) :-
        input_lunchHourList(Slots),
        length(Slots, LastIndex),
        nth1(LastIndex, Slots, End).

countLunchSlots(Number) :-
        input_lunchHourList(Slots),
        length(Slots, Number).

% ******************** Partial Workers ***********************

partialSalaryBase(Salary) :-
        input_partialSalaryPerHour(PerHour),
        input_partialWorkHours(Hours),
        Salary is PerHour * Hours.

partialSalaryPlus(Salary) :-
        partialSalaryBase(SalaryBase),
        input_partialBonus(Bonus),
        Salary is SalaryBase + Bonus.


% ************************************************************
% ************************** DOMAINS *************************
% ************************************************************

% ************************************************************
% ***************** Full Time ****************

% Defines the lunch hours list domain for Full Workers: -2: doesn't work, -1 works at lunch hour
defineLunchHoursDomain([]).
defineLunchHoursDomain([Hour|HourRest]) :-
        startLunchHour(Start),
        endLunchHour(End),

        Hour in {-2} \/ {-1} \/ (Start..End),
        defineLunchHoursDomain(HourRest).

% Defines the domain of the possible salaries:
defineFullWorkersSalary([]).
defineFullWorkersSalary([Salary|RSalary]) :-
        fullSalaryBase(SalaryBase),
        fullSalaryPlus(SalaryPlus),

        Salary in {0} \/ {SalaryBase} \/ {SalaryPlus},
        defineFullWorkersSalary(RSalary).

% Defines the domains for the Full Time Decision variables
defineFullDomains(FullWorkersLunchHours, FullWorkersLunching, FullWorkersSalary, NumberFullWorkers, MaxFullWorkers, Slots) :-

        countLunchSlots(NumberLunchSlots), % Input

        % The maximum number of workers possible is 2 times the maximum input number:
        max_member(MaxFullWorkersSlots, Slots),
        MaxFullWorkers is 2 * MaxFullWorkersSlots,

        % Decision variables declaration:
        length(FullWorkersLunchHours, MaxFullWorkers), % Main variable
        length(FullWorkersLunching, NumberLunchSlots), % Auxiliar:
        length(FullWorkersSalary, MaxFullWorkers),

        % Domain definition:
        defineLunchHoursDomain(FullWorkersLunchHours),
        domain(FullWorkersLunching, 0, MaxFullWorkers),
        defineFullWorkersSalary(FullWorkersSalary),
        NumberFullWorkers in 0..MaxFullWorkers.


```

```
% ************************************************************
% *************** Part time ****************

% Defines the domain for the start hour of the partial worker shift
definePartialHoursDomain([]).
definePartialHoursDomain([Hour|HourRest]) :-
        input_startWork(Start),
        input_endWork(End),

        Hour in {-1} \/ (Start..End),
        definePartialHoursDomain(HourRest).

% Defines the domain of the possible salaries:
definePartialWorkersSalary([]).
definePartialWorkersSalary([Salary|RSalary]) :-
        partialSalaryBase(SalaryBase),
        partialSalaryPlus(SalaryPlus),

        Salary in {0} \/ {SalaryBase} \/ {SalaryPlus},
        definePartialWorkersSalary(RSalary).

% Defines the domains for the Partial Time Decision variables
definePartialDomains(PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary, PartialWorkersPerSlot,
    PartialWorkersExists, Slots) :-
        totalWorkHours(NumberSlots),
        input_partialMaxWorkers(MaxPartialWorkersRes),

        % The maximum number of workers possible is minimum(2 times the maximum slot number, rescrited value from the input):
        max_member(MaxPartialWorkersSlots, Slots),
        MaxPartialWorkersInput is 2 * MaxPartialWorkersSlots,
        min_member(MaxPartialWorkers, [MaxPartialWorkersInput, MaxPartialWorkersRes]),

        % Decision variables declaration:
        length(PartialWorkersStartHour, MaxPartialWorkers), % Main variables
        length(PartialWorkersExtraHour, MaxPartialWorkers),
        length(PartialWorkersSalary, MaxPartialWorkers), % Auxiliar
        length(PartialWorkersPerSlot, NumberSlots),
        length(PartialWorkersExists, MaxPartialWorkers),

        % Domain definition:
        definePartialHoursDomain(PartialWorkersStartHour),
        domain(PartialWorkersExtraHour, 0, 1),
        definePartialWorkersSalary(PartialWorkersSalary),
        domain(PartialWorkersPerSlot, 0, MaxPartialWorkers),
        domain(PartialWorkersExists, 0, 1).

% ************************************************************************
% ***************************** RESTRICTIONS *****************************
% ************************************************************************

% ************************************************************
% *************** Full Time Workers Restrictions ***************

% Applies the restrictions to the full workers decision variables
applyFullRestrictions(FullWorkersLunchHours, FullWorkersLunching, FullWorkersSalary, NumberFullWorkers, MaxFullWorkers,
    NoLunchOut) :-
        input_lunchHourList(LunchHours),

        fullSalaryBase(SalaryBase), fullSalaryPlus(SalaryPlus),

        % Ensure order on the lunch hours (avoid symmetric solutions):
        restrictOrderedList(FullWorkersLunchHours),

        % Restrict the salary according to the value in the lunch hour (also get non-Workers and non-Lunch workers):
        restrictFullHours(FullWorkersLunchHours, FullWorkersSalary),
        global_cardinality(FullWorkersSalary, [0-NoWorkersCount, SalaryBase-_, SalaryPlus-NoLunchOut]),

        % Count the number of full workers on each lunching slot:
```

```
        countWorkersLunching(LunchHours, FullWorkersLunchHours, FullWorkersLunching),

        % Restrict the number of real workers:
        NumberFullWorkers #= MaxFullWorkers - NoWorkersCount.

% Restricts the full workers salary according to FullWorkerLunchHour decision variable (also count non worker and non lunch workers)
restrictFullHours([], []).
restrictFullHours([FullWorkerLunchHour|RFullWorkerLunchHour], [Salary|RSalary]) :-
        fullSalaryBase(SalaryBase), fullSalaryPlus(SalaryPlus),

        (FullWorkerLunchHour #= -2 #/\ Salary #= 0) #\/
        (FullWorkerLunchHour #= -1 #/\ Salary #= SalaryPlus) #\/
        (FullWorkerLunchHour #>= 0 #/\ Salary #= SalaryBase),

        restrictFullHours(RFullWorkerLunchHour, RSalary).

% Counts the number of full workers lunching on each of the possible lunch hours
countWorkersLunching([], _, []).
countWorkersLunching([LunchHour|RLunchHours], FullWorkersLunchHours, [FullWorkersLunching|RFullWorkersLunching]) :-
        countWorkersLunchingAux(LunchHour, FullWorkersLunchHours, FullWorkersLunching),

        countWorkersLunching(RLunchHours, FullWorkersLunchHours, RFullWorkersLunching).

countWorkersLunchingAux(_, [], 0).
countWorkersLunchingAux(LunchHour, [FullWorkerLunchHour|RFullWorkerLunchHour], FullWorkersLunchingOut) :-
        (FullWorkerLunchHour #= LunchHour) #<=> Bool,
        FullWorkersLunchingOut #= FullWorkersLunchingNext + Bool,

        countWorkersLunchingAux(LunchHour, RFullWorkerLunchHour, FullWorkersLunchingNext).


% ********************************************************************
% ***************** Partial Time Workers Restrictions ****************

% Applies the restrictions to the partial workers decision variables
applyPartialRestrictions(PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary, PartialWorkersPerSlot, PartialWorkersExists) :-
        input_startWork(StartHour),

        % Ensure order on the start hours:
        restrictOrderedList(PartialWorkersStartHour),

        restrictPartialHours(PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary, PartialWorkersExists),
        restrictPartialSlots(StartHour, PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersPerSlot).

% Restricts the full workers salary (and PartialWorkersExist) according to the StartHour and ExtraHour decision variable
restrictPartialHours([], [], [], []).
restrictPartialHours([StartHour|RStartHour], [ExtraHour|RExtraHour], [Salary|RSalary], [PartialWorkersExists|RPartialWorkersExits]) :-
        partialSalaryBase(SalaryBase), partialSalaryPlus(SalaryPlus),

        (StartHour #> -1 #/\ ExtraHour #= 0 #/\ Salary #= SalaryBase #/\ PartialWorkersExists #= 1) #\/
        (StartHour #> -1 #/\ ExtraHour #= 1 #/\ Salary #= SalaryPlus #/\ PartialWorkersExists #= 1)  #\/
        (StartHour #= -1 #/\ ExtraHour #= 0 #/\ Salary #= 0 #/\ PartialWorkersExists #= 0),

        restrictPartialHours(RStartHour, RExtraHour, RSalary, RPartialWorkersExits).

% Counts the number of partial workers on each Time Slot
restrictPartialSlots(_, _, _, []).
restrictPartialSlots(StartHour, PartialWorkersStartHour, PartialWorkersExtraHour, [PartialWorkersPerSlot|RPartialWorkersPerSlot]) :-
        restrictPartialSlotsAux(StartHour, PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersPerSlot),

        NextHour is StartHour + 1,
        restrictPartialSlots(NextHour, PartialWorkersStartHour, PartialWorkersExtraHour, RPartialWorkersPerSlot).

restrictPartialSlotsAux(_, [], [], 0).
```

```
restrictPartialSlotsAux(CurrentHour, [StartHour|RStartHour], [ExtraHour|RExtraHour], PartialWorkersPerSlotOut) :-
        input_partialWorkHours(MaxHours),

        ((CurrentHour #>= StartHour #/\ CurrentHour #=< StartHour + MaxHours - 1 #/\ ExtraHour #= 0) #\/
        (CurrentHour #>= StartHour #/\ CurrentHour #=< StartHour + MaxHours #/\ ExtraHour #= 1)) #<=> Number,
        PartialWorkersPerSlotOut #= Number + NumberTemp,

        restrictPartialSlotsAux(CurrentHour, RStartHour, RExtraHour, NumberTemp).


% ************************************************************
% ***************** Common Workers Restrictions ****************

% Ensures that the list is ordered (used to avoid symmetric solutions)
restrictOrderedList(List) :-
        length(List, ListSize),
        length(Xs, ListSize),
        length(Ps, ListSize),

        sorting(Xs,Ps,List).

% Applies the restrictions to the full/partial workers decision variables
applyCommonRestrictions(NoLunch, FullWorkersLunching, NumberFullWorkers, PartialWorkersExtraHour, PartialWorkersPerSlot,
     PartialWorkersExists, Slots, NumberPartialWorkers) :-
        input_startWork(StartHour),
        input_lunchHourList(LunchHours),
        input_maxExtraWorkers(MaxExtraWorkers),
        input_partialWorkersRatio(PartialRatio),

        % Restrict the number of full workers with no lunch + partial workers with extra hour:
        sum(PartialWorkersExtraHour, #=, TotalExtraHour),
        TotalExtraHour + NoLunch #=< MaxExtraWorkers,

        % Restrict the ratio of full/partial workers:
        sum(PartialWorkersExists, #=, NumberPartialWorkers),
        NumberPartialWorkers #=< (PartialRatio * NumberFullWorkers) // 100, % HARCODED

        % Restricts the number of total workers on each Time Slot (has to be at minimum the original Time Slot)
        restrictSlots(StartHour, LunchHours, FullWorkersLunching, NumberFullWorkers, PartialWorkersPerSlot, Slots).


restrictSlots(_, [], [], _, [], []).

% If lunch time, restrict current slot and advance all lists:
restrictSlots(CurrentHour, [LunchHour|RLunchHours], [FullWorkersLunching|RFullWorkersLunching], NumberFullWorkers,
     [PartialWorkersPerSlot|RPartialWorkersPerSlot], [CurrentSlot|RSlots]) :-
        CurrentHour = LunchHour,
        CurrentSlot #=< (NumberFullWorkers - FullWorkersLunching) + PartialWorkersPerSlot,

        NextHour is CurrentHour + 1,
        restrictSlots(NextHour, RLunchHours, RFullWorkersLunching, NumberFullWorkers, RPartialWorkersPerSlot, RSlots).

% If not lunch time, restrict current slot and advance PartialWorkersPerSlot list and Slots list:
restrictSlots(CurrentHour, LunchHours, FullWorkersLunching, NumberFullWorkers, [PartialWorkersPerSlot|RPartialWorkersPerSlot],
     [CurrentSlot|RSlots]) :-
        CurrentSlot #=< NumberFullWorkers + PartialWorkersPerSlot,

        NextHour is CurrentHour + 1,
        restrictSlots(NextHour, LunchHours, FullWorkersLunching, NumberFullWorkers, RPartialWorkersPerSlot, RSlots).

% ************************************************************
% ***************** Solution Cost Calculation ******************
calculateCost(FullWorkersSalary, PartialWorkersSalary, TotalSalary) :-
        sum(FullWorkersSalary, #=, TotalFullWorkersSalary),
        sum(PartialWorkersSalary, #=, TotalPartialWorkersSalary),
        TotalSalary #= TotalFullWorkersSalary + TotalPartialWorkersSalary.

% ************************************************************
```

```
% ********************** Solution Search **********************
searchSolution(FullWorkersLunchHours, PartialWorkersStartHour, PartialWorkersExtraHour, TotalSalary) :-
        append([FullWorkersLunchHours, PartialWorkersStartHour, PartialWorkersExtraHour], Vars),

        write('Labeling...'),
        labeling([ffc, step, up, minimize(TotalSalary)], Vars).


% ************************************************************
% ********************** Final Slot Calculation ***************

calculateFinalSolution(FullWorkersLunching, NumberFullWorkers, PartialWorkersPerSlot, FinalSlots) :-
        totalWorkHours(SlotNumber),
        input_startWork(StartHour),
        input_lunchHourList(LunchSlots),

        length(FinalSlots, SlotNumber),

        calculateFinalSlots(StartHour, LunchSlots, FullWorkersLunching, NumberFullWorkers, PartialWorkersPerSlot, FinalSlots).

calculateFinalSlots(_, [], [], _, [], []).
calculateFinalSlots(CurrentHour, [LunchHour|RLunchHours], [FullWorkersLunching|RFullWorkersLunching], NumberFullWorkers,
    [PartialWorkersPerSlot|RPartialWorkersPerSlot], [FinalSlot|RFinalSlots]) :-
        CurrentHour = LunchHour,
        FinalSlot is NumberFullWorkers - FullWorkersLunching + PartialWorkersPerSlot,

        NextHour is CurrentHour + 1,
        calculateFinalSlots(NextHour, RLunchHours, RFullWorkersLunching, NumberFullWorkers, RPartialWorkersPerSlot, RFinalSlots).

calculateFinalSlots(CurrentHour, LunchHours, FullWorkersLunching, NumberFullWorkers,
    [PartialWorkersPerSlot|RPartialWorkersPerSlot], [FinalSlot|RFinalSlots]) :-
        FinalSlot is NumberFullWorkers + PartialWorkersPerSlot,

        NextHour is CurrentHour + 1,
        calculateFinalSlots(NextHour, LunchHours, FullWorkersLunching, NumberFullWorkers, RPartialWorkersPerSlot, RFinalSlots).




% ************************************************************
% ******************* MAIN SOLVER ****************************
% ************************************************************

schedule :-
        prepareStatistics,

        input_slots(SlotsInput),

        % Define decision variables domains:
        defineFullDomains(FullWorkersLunchHours, FullWorkersLunching, FullWorkersSalary, NumberFullWorkers, MaxFullWorkers,
    SlotsInput),
        definePartialDomains(PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary, PartialWorkersPerSlot,
    PartialWorkersExists, SlotsInput),

        % Apply restrictions:
        applyFullRestrictions(FullWorkersLunchHours, FullWorkersLunching, FullWorkersSalary, NumberFullWorkers, MaxFullWorkers,
    NoLunch),
        applyPartialRestrictions(PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary, PartialWorkersPerSlot,
    PartialWorkersExists),
        applyCommonRestrictions(NoLunch, FullWorkersLunching, NumberFullWorkers, PartialWorkersExtraHour, PartialWorkersPerSlot,
    PartialWorkersExists, SlotsInput, NumberPartialWorkers),

        % Calculate solution cost:
        calculateCost(FullWorkersSalary, PartialWorkersSalary, TotalSalary),

        % Label variables according to the best solution possible and calculate the final slots:
        searchSolution(FullWorkersLunchHours, PartialWorkersStartHour, PartialWorkersExtraHour, TotalSalary),
        calculateFinalSolution(FullWorkersLunching, NumberFullWorkers, PartialWorkersPerSlot, FinalSlots),
```

```
353
354         % Show the time passed:
355         showStatistics,
356
357         % Debug:
358         %write('Full Workers Lunch Hours: '), write(FullWorkersLunchHours), nl,
359         %write('Full Workers Lunching: '), write(FullWorkersLunching), nl,
360         %write('Full Workers Salary: '), write(FullWorkersSalary), nl,
361         %write('NumberFullWorkers: '), write(NumberFullWorkers), nl,
362         %write('Partial Workers Start Hour: '), write(PartialWorkersStartHour), nl,
363         %write('Partial Workers Extra Hour: '), write(PartialWorkersExtraHour), nl,
364         %write('Partial Workers Salary: '), write(PartialWorkersSalary), nl,
365         %write('Partial Workers Per Slot: '), write(PartialWorkersPerSlot), nl,
366
367         % **** Display optimal solution: ****
368         printSolution(FinalSlots, NoLunch, NumberFullWorkers, FullWorkersLunching,
369                       PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary, NumberPartialWorkers,
370                       TotalSalary).
371
372
373 %slotsEx(Slots) :- Slots = [4, 3, 3, 6, 5, 6, 8, 8].
374 %slotsEx2(Slots) :- Slots = [2, 2, 2, 2, 2, 2, 2, 2].
375 %slotsEx3(Slots) :- Slots = [3, 3, 3, 3, 3, 3, 3, 3].
376 %slotsBig(Slots) :- Slots = [100, 100, 100, 100, 100, 100, 100, 100].
377 %slotsBig2(Slots) :- Slots = [40, 50, 40, 70, 50, 60, 30, 20].
378
379
380 % ************************************************************************
381 % ******************************** PRINT *********************************
382 % ************************************************************************
383
384 % Prints the solution: textual, final timetable and graphical solution:
385 printSolution(FinalSlots, NoLunch, NumberFullWorkers, FullWorkersLunching, PartialWorkersStartHour, PartialWorkersExtraHour,
        PartialWorkersSalary, NumberPartialWorkers, TotalSalary) :-
386         input_lunchHourList(LunchSlots),
387         input_startWork(StartHour),
388         input_endWork(EndHour),
389
390         write('*** Solution ***'), nl, nl,
391
392         % Print textual information regarding the full and partial workers shifts:
393         write('* Textual display of the schedule for each type of worker: '), nl,
394         printFullWorkers(LunchSlots, NoLunch, NumberFullWorkers, FullWorkersLunching), nl,
395         printPartialWorkers(PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary, NumberPartialWorkers), nl,
396
397         printTimetable(StartHour, EndHour, FinalSlots), nl, nl,
398         printScheduleBoard(StartHour, EndHour, LunchSlots, FullWorkersLunching, NoLunch, PartialWorkersStartHour,
        PartialWorkersExtraHour, PartialWorkersSalary), nl,
399
400         write('Total Salary Cost: '), write(TotalSalary), nl.
401
402 % ***********************************************
403 % ************** Textual Solution ***************
404 % ***********************************************
405 printFullWorkers(LunchHours, NoLunch, NumberFullWorkers, FullWorkersLunching) :-
406         fullSalaryPlus(SalaryPlus),
407
408         write(NumberFullWorkers), write(' full workers'), nl,
409         write('-> '), write(NoLunch), write(' full workers do extra hour at lunch and receive a salary of '), write(SalaryPlus),
        nl,
410
411         printFullWorkersLunching(LunchHours, FullWorkersLunching).
412
413 printFullWorkersLunching([], []).
414 printFullWorkersLunching([LunchHour|RLunchHour], [FullWorkersLunching|RFullWorkersLunching]) :-
415         fullSalaryBase(SalaryBase),
416
```

```
        write('-> '), write(FullWorkersLunching), write(' full workers lunch at '), write(LunchHour), write(' and receive a salary of '), write(SalaryBase), nl,
        printFullWorkersLunching(RLunchHour, RFullWorkersLunching).

printPartialWorkers(PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary, NumberPartialWorkers) :-
        write(NumberPartialWorkers), write(' partial workers'), nl,
        printPartialWorkersAux(PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary).

printPartialWorkersAux([], [], []).
printPartialWorkersAux([StartHour|RStartHour], [ExtraHour|RExtraHour], [Salary|RSalary]) :-
        StartHour > -1,
        calculateEndHour(StartHour, ExtraHour, EndHour),
        write('-> A partial worker starts at '), write(StartHour), write(', ends at '), write(EndHour), write(' and receives a salary of '), write(Salary), nl,
        printPartialWorkersAux(RStartHour, RExtraHour, RSalary).
printPartialWorkersAux([_|RStartHour], [_|RExtraHour], [_|RSalary]) :-
        printPartialWorkersAux(RStartHour, RExtraHour, RSalary).


calculateEndHour(StartHour, 0, EndHourOut) :-
        input_partialWorkHours(Hours),
        EndHourOut is StartHour + Hours.
calculateEndHour(StartHour, 1, EndHourOut) :-
        input_partialWorkHours(Hours),
        EndHourOut is StartHour + Hours + 1.


% ***********************************************
% ************** Final Timetable ***************
% ***********************************************
printTimetable(StartHour, LastHour, Slots) :-
        nl, write('* Final timetable: '), nl,
        write('Hours      '), printHours(StartHour, LastHour), nl,
        write('Workers    '), printSlots(StartHour, LastHour, Slots), nl.

printHours(CurrentHour, LastHour) :-
        CurrentHour =< LastHour,
        format("~5|~d~5+", [CurrentHour]),

        NextHour is CurrentHour + 1,
        printHours(NextHour, LastHour).
printHours(_, _).

printSlots(_, _, []).
printSlots(CurrentHour, LastHour, [CurrentSlot|RSlots]) :-
        CurrentHour =< LastHour,
        format("~5|~d~5+", [CurrentSlot]),

        NextHour is CurrentHour + 1,
        printSlots(NextHour, LastHour, RSlots).

% ***********************************************
% ************** Schedual Board ****************
% ***********************************************
printScheduleBoard(StartHour, LastHour, LunchHours, FullWorkersLunching, NoLunch, PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary) :-
        fullSalaryBase(SalaryBase),
        fullSalaryPlus(SalaryPlus),

        write('* Graphical solution of all the workers\' schedule:'), nl,

        printHours(StartHour, LastHour), write('Salary'), nl,
        printFullWorkersLunching(StartHour, LastHour, LunchHours, FullWorkersLunching, SalaryBase),
        printFullWorkersNoLunch(StartHour, LastHour, NoLunch, SalaryPlus),
        printPartialWorkersX(StartHour, LastHour, PartialWorkersStartHour, PartialWorkersExtraHour, PartialWorkersSalary).

printFullWorkersLunching(_, _, [], [], _).
printFullWorkersLunching(StartHour, LastHour, [LunchHour|RLunchHour], [FullWorkersLunching|RFullWorkersLunching], Salary) :-
```

```
        printFullWorkersAtLunchX(StartHour, LastHour, LunchHour, FullWorkersLunching, Salary),
        printFullWorkersLunching(StartHour, LastHour, RLunchHour, RFullWorkersLunching, Salary).

printFullWorkersAtLunchX(_, _, _, 0, _).
printFullWorkersAtLunchX(StartHour, LastHour, LunchHour, Counter, Salary) :-
        printFullWorkerAtLunchX(StartHour, LastHour, LunchHour), format("~5|~d~5+", [Salary]), nl,

        NextCounter is Counter - 1,
        printFullWorkersAtLunchX(StartHour, LastHour, LunchHour, NextCounter, Salary).

printFullWorkerAtLunchX(CurrentHour, LastHour, LunchHour) :-
        CurrentHour =< LastHour,
        CurrentHour = LunchHour,
        format("~5|~s~5+", [" "]),

        NextHour is CurrentHour + 1,
        printFullWorkerAtLunchX(NextHour, LastHour, LunchHour).
printFullWorkerAtLunchX(CurrentHour, LastHour, LunchHour) :-
        CurrentHour =< LastHour,
        format("~5|~s~5+", ["X"]),

        NextHour is CurrentHour + 1,
        printFullWorkerAtLunchX(NextHour, LastHour, LunchHour).
printFullWorkerAtLunchX(_, _, _).

printFullWorkersNoLunch(_, _, 0, _).
printFullWorkersNoLunch(StartHour, LastHour, NoLunch, Salary) :-
        printFullWorkerExtraHour(StartHour, LastHour), format("~5|~d~5+", [Salary]), nl,

        NextNoLunch is NoLunch - 1,
        printFullWorkersNoLunch(StartHour, LastHour, NextNoLunch, Salary).

printFullWorkerExtraHour(CurrentHour, LastHour) :-
        CurrentHour =< LastHour,
        format("~5|~s~5+", ["X"]),

        NextHour is CurrentHour + 1,
        printFullWorkerExtraHour(NextHour, LastHour).
printFullWorkerExtraHour(_, _).

printPartialWorkersX(_, _, [], [], []).
printPartialWorkersX(StartHour, LastHour, [PartialWorkersStartHour|RPartialWorkersStartHour],
    [PartialWorkersExtraHour|RPartialWorkersExtraHour], [Salary|RSalary]) :-
        PartialWorkersStartHour >= 0,
        printPartialWorkerX(StartHour, LastHour, PartialWorkersStartHour, PartialWorkersExtraHour), format("~5|~d~5+", [Salary]),
    nl,
        printPartialWorkersX(StartHour, LastHour, RPartialWorkersStartHour, RPartialWorkersExtraHour, RSalary).
printPartialWorkersX(StartHour, LastHour, [_|RPartialWorkersStartHour], [_|RPartialWorkersExtraHour], [_|RSalary]) :-
        printPartialWorkersX(StartHour, LastHour, RPartialWorkersStartHour, RPartialWorkersExtraHour, RSalary).

printPartialWorkerX(CurrentHour, LastHour, PartialWorkerStartHour, PartialWorkerExtraHour) :-
        input_partialWorkHours(PartialHours),

        CurrentHour =< LastHour,
        CurrentHour >= PartialWorkerStartHour,
        PartialWorkerExtraHour = 0,
        CurrentHour =< PartialWorkerStartHour + PartialHours - 1,
        format("~5|~s~5+", ["X"]),

        NextHour is CurrentHour + 1,
        printPartialWorkerX(NextHour, LastHour, PartialWorkerStartHour, PartialWorkerExtraHour).
printPartialWorkerX(CurrentHour, LastHour, PartialWorkerStartHour, PartialWorkerExtraHour) :-
        input_partialWorkHours(PartialHours),

        CurrentHour =< LastHour,
        CurrentHour >= PartialWorkerStartHour,
        PartialWorkerExtraHour = 1,
        CurrentHour =< PartialWorkerStartHour + PartialHours,
```

```
        format("~5|~s~5+", ["X"]),

        NextHour is CurrentHour + 1,
        printPartialWorkerX(NextHour, LastHour, PartialWorkerStartHour, PartialWorkerExtraHour).
printPartialWorkerX(CurrentHour, LastHour, PartialWorkerStartHour, PartialWorkerExtraHour) :-
        CurrentHour =< LastHour,
        format("~5|~s~5+", [" "]),

        NextHour is CurrentHour + 1,
        printPartialWorkerX(NextHour, LastHour, PartialWorkerStartHour, PartialWorkerExtraHour).
printPartialWorkerX(_, _, _, _).


% ************************************************************
% ************************ STATISTICS ************************
% ************************************************************

prepareStatistics :-
        fd_statistics(resumptions, _),
        fd_statistics(entailments, _),
        fd_statistics(prunings, _),
        fd_statistics(backtracks, _),
        fd_statistics(constraints, _),
        statistics(total_runtime, _).

showStatistics :-
        nl, nl, write('*** Sicstus: Statistics ***'), nl,
        %fd_statistics,
        statistics(total_runtime,[_,Time]),
        write('Time passed (ms): '),  write(Time), nl, nl, nl.
```