Restrições - Satisfação ou Otimização

- Em certos (muitos) casos o que se pretende determinar não é uma solução qualquer, mas sim a melhor solução.
- Neste caso, o problema é mais adequadamente descrito como um problema de otimização com restrições.
- Um problema de otimização com restrições pode ser especificado por um tuplo < V, D, R, F > em que
  - **V**: é o conjunto de variáveis usadas na modelação do problema
  - **D**: é o domínio(s) em que as variáveis de V podem tomar valores
  - **R**: é o conjunto de restrições que afectam as variáveis V
  - **F**: é uma função das soluções para um domínio ordenado

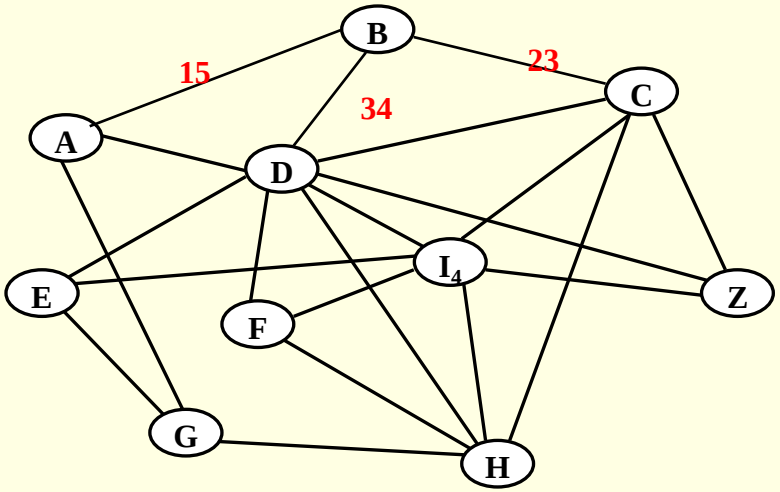
Otimização com Restrições - Exemplos

- Planeamento de testes em circuitos digitais
  - Teste com menos entradas especificadas
- Gestão de Tráfego em Redes
  - Tráfego com menor custo
  - Máximo tráfego com um dado custo
- Gestão da Produção
  - Plano com maior lucro
  - Máxima produção com os recursos existentes
- Sequenciação de Tarefas (Scheduling)
  - Solução com o fim mais cedo
  - Ocupação máxima das máquinas existentes

Otimização com Restrições - Exemplos

- Geração de Horários
  - Solução com menos furos
  - Solução com menos teóricas de tarde
- Caixeiro Viajante
  - Solução com menor distância percorrida
- “Colocação” ou “preenchimento”
  - Colocação do máximo número de peças

Restrições + Otimização - Caixeiro Viajante



- Com as definições anteriores e denotando por d_{ij} a distância efetivamente percorrida entre os nós i e j,
  - Definição da distância percorrida

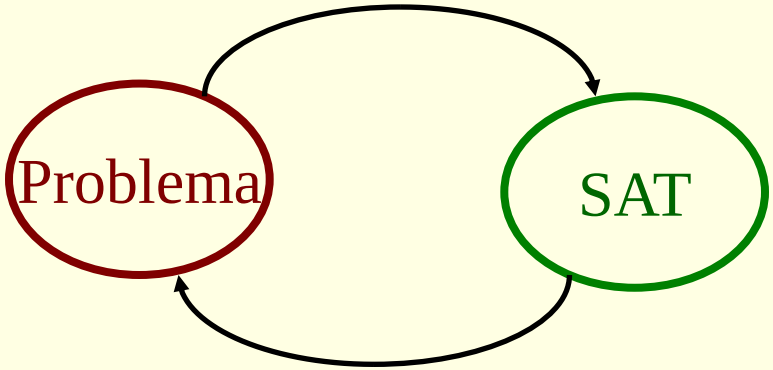
$$|V_i - V_j| = 1 \Rightarrow c_{ij} < M$$
$$|V_i - V_j| = 1 \Rightarrow d_{ij} = c_{ij}$$
$$|V_i - V_j| \neq 1 \Rightarrow d_{ij} = 0$$

  - Otimização

$$\min \Sigma_i \Sigma_j d_{ij}$$

Restrições – Complexidade

- Os problemas de interesse de satisfação de restrições são geralmente NP – completos:
  - Descoberta de solução: complexidade exponencial (pior caso)
  - Verificação de solução: complexidade polinomial
- Já os problemas otimização de restrições são geralmente NP – difíceis:
  - Descoberta de solução: complexidade exponencial (pior caso)
  - Verificação de solução: complexidade exponencial
- Analogia com SAT



Optimisation

Although Constraint Logic Programming is somehow focused in constraint satisfaction (closer to a "logical" view), constraint optimisation is usually a very important goal — often, "the" goal that a user expects from an application.

Example:

For all the scheduling problems that were considered, one could aim at optimising some goal, namely

- To obtain the earliest end time for all the tasks, with limited resources; or
- The solution that requires less resources to complete at a certain due time.

Optimisation

Optimisation has been extensively studied for the case of linear constraints over the rational/real numbers.

For these problems fast algorithms exist, based not only in the Simplex algorithm and all its variants, but also in interior point methods. Software packages are also available, incorporating such algorithms (e.g. CPLEX).

In the context of finite domains, it is important to consider situations where

- Variables may take values on a finite set of (integer) values;
- The constraints to satisfy are not linear;
- The optimisation function is not linear.

Optimisation

The computation of optimal solutions may be performed by many algorithms taking either

- a constructive approach; or
- a repairing approach.

In the constructive approach, the optimisation process may be regarded as the progressive instantiation of the variables that model the problem. The goal is to obtain the best complete solution, i.e. one that can be proved that is better than any other.

In the repairing approach, a complete solution is iteratively being changed into some of its "neighbours". Again the goal is to reach an optimal solution, but these methods do not usually guarantee the optima.

Optimisation

The constructive approach to optimisation uses in general two types of techniques

- Search
- Relaxation

These two concepts are ilustrated in the following BIP (Binary Integer Programming) problem

$$\begin{aligned} \text{Max } Z &= 9x_1 + 5x_2 + 6x_3 + 4x_4 \\ \text{Subj.} \quad & 6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10 \\ & - x_3 + 2x_4 \leq 10 \\ & -x_1 + x_3 \leq 0 \\ & - x_2 + x_4 \leq 0 \end{aligned}$$

Optimisation

$$\begin{aligned} \text{Max } Z &= 9x_1 + 5x_2 + 6x_3 + 4x_4 \\ \text{Subj.} \quad & 6x_1 + 3x_2 + 5x_3 + 2x_4 \le 10 \\ & - x_3 + 2x_4 \le 10 \\ & -x_1 + x_3 \le 0 \\ & - x_2 + x_4 \le 0 \end{aligned}$$

Since the variables are binary, X1 = 0 / 1, the problem may be decomposed in two separate problems.

Search is required to find the overall best solution within the two smaller problems.

$$\begin{aligned} \text{Max } & 5x_2 + 6x_3 + 4x_4 \\ \text{Sbj } & 3x_2 + 5x_3 + 2x_4 \le 10 \\ & -x_3 + 2x_4 \le 1 \\ & x_3 \le 0 \\ & -x_2 + x_4 \le 0 \end{aligned}$$

$$\begin{aligned} \text{Max } & 9+5x_2 + 6x_3 + 4x_4 \\ \text{Sbj } & 3x_2 + 5x_3 + 2x_4 \le 4 \\ & - x_3 + 2x_4 \le 1 \\ & x_3 \le 1 \\ & -x_2 + x_4 \le 0 \end{aligned}$$

Optimisation

In general, relaxation aims at simplifying a problem (making it less constrained) such that

- the simplified problem is easier to solve
- the solutions found are "informative".

In this context, informative means that it might be possible to "infer" properties of the solutions of the initial problem, based on the solutions of the simplified problem.

Since the integers are a subset of the rationals (or of the reals), a common relaxation with linear constraints consists of dropping the integrality constraint on the variables domains, and use techniques appicable to the real numbers.

Optimisation

In this case, the relaxation of variables 0/1 consists of simply considering them in the interval [0..1].

The two subproblems may now be handled as linear programming problems, possibly by some of the fast tools available, providing the following optima

$Z_{max} = 9$ w/(0,1,0,1) $Z_{max}=16.2$ w/(1,.8,0,.8)

$$\begin{aligned} \text{Max } Z &= 5x_2+6x_3+4x_4 \\ \text{Sbj } 3x_2+ 5x_3+ 2x_4 &\le 10 \\ -x_3+ 2x_4 &\le 1 \\ x_3 &\le 0 \\ -x_2 + x_4 &\le 0 \end{aligned} \qquad \begin{aligned} \text{Max } Z &= 9+5x_2+6x_3+4x_4 \\ \text{Sbj } 3x_2+ 5x_3+ 2x_4 &\le 4 \\ -x_3+ 2x_4 &\le 1 \\ x_3 &\le 1 \\ -x_2 + x_4 &\le 0 \end{aligned}$$

Search looks then more promising within the subproblem for which X1 =1

Optimisation

The work done so far has shown that:

Problem X1 = 0 has an exact (integer) solution. Hence problems should only be exploited if they have the potential to improve this value, that may be considered the best solution (for the whole problem) so far.

Problem X1 = 1 has a relaxed solution of 16.2. Since there is here the potential to find an integer solution better than the best so far, this is the problem to search next.

$$Z_{max} = 16.2 \text{ w/}(1,.8,0,.8)$$

$$\text{Max } Z = 9 + 5x_2 + 6x_3 + 4x_4$$

$$\begin{aligned} \text{Sbj } & 3x_2 + 5x_3 + 2x_4 \leq 4 \\ & -x_3 + x_4 \leq 1 \\ & x_3 \leq 1 \\ & -x_2 + x_4 \leq 0 \end{aligned}$$

Optimisation

Again decomposing the problem through variable X2:

Z_{max}= 16.2 c/(1,.8,0,.8)

$$\begin{aligned} \text{Max } Z &= 9 + 5x_2 + 6x_3 + 4x_4 \\ \text{Sbj } & 3x_2 + 5x_3 + 2x_4 \le 4 \\ & -x_3 + x_4 \le 1 \\ & x_3 \le 1 \\ & -x_2 + x_4 \le 0 \end{aligned}$$

Z_{max}= 13.8 (1,0,.8,0)

$$\begin{aligned} \text{Max } Z &= 9 + 6x_3 + 4x_4 \\ \text{Sbj } & 5x_3 + 2x_4 \le 4 \\ & -x_3 + x_4 \le 1 \\ & x_3 \le 0 \\ & x_4 \le 0 \end{aligned}$$

Z_{max}= 16 c/(1,1,0,.5)

$$\begin{aligned} \text{Max } Z &= 14 + 6x_3 + 4x_4 \\ \text{Sbj } & 5x_3 + 2x_4 \le 1 \\ & -x_3 + x_4 \le 1 \\ & x_3 \le 1 \\ & x_4 \le 1 \end{aligned}$$

Optimisation

None of the solutions is integer. For showing best potential we continue with problem (exploiting X3=0 and X3=1):

Z_{max}= 16 (1,1,0,.5)

$$\begin{aligned} \text{Max } & 14 + 6x_3 + 4x_4 \\ \text{Sbj } & 5x_3 + 2x_4 \le 1 \\ & -x_3 + x_4 \le 1 \\ & x_3 \le 1 \\ & x_4 \le 1 \end{aligned}$$

Z_{max}= 16 (1,1,0,.5)

$$\begin{aligned} \text{Max } Z = & 14 + 4x_4 \\ \text{Sbj } & 2x_4 \le 1 \\ & x_4 \le 1 \\ & 0 \le 0 \\ & x_4 \le 1 \end{aligned}$$

Z_{max}= - (1,1,1,-)

$$\begin{aligned} \text{Max } Z = & 20 + 4x_4 \\ \text{Sbj } & 2x_4 \le -4 \\ & x_4 \le 2 \\ & 1 \le 1 \\ & x_4 \le 1 \end{aligned}$$

Optimisation

The problem that resulted from X3 = 1 eventually led to the dissatisfaction of one of the constraints.

$$\begin{aligned} \textbf{Max } & Z = 20 + 4x_4 \\ \textbf{Sbj } & 2x_4 \leq -4 \\ & x_4 \leq 2 \\ & 1 \leq 1 \\ & x_4 \leq 1 \end{aligned}$$

This problem can be then simply pruned from the search, and not considered any longer.

Optimisation

We may then proceed with the other problem, where three variables are already instantiated (X1=1, X2=1 e X3=0) :

$Z_{max} = 16(1,1,0,.5)$

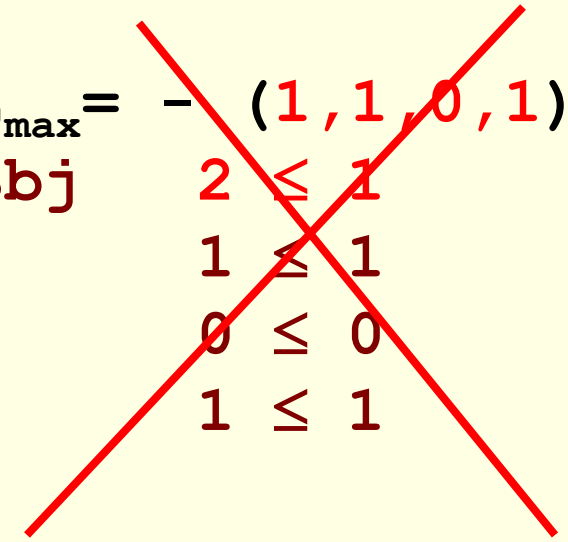
$$\begin{aligned} \text{Max } Z &= 14 + 4x_4 \\ \text{Sbj } 2x_4 &\leq 1 \\ x_4 &\leq 1 \\ 0 &\leq 0 \\ x_4 &\leq 1 \end{aligned}$$

$Z_{max} = 14\ (1,1,0,0)$

$$\begin{aligned} \text{Sbj } 0 &\leq 1 \\ 0 &\leq 1 \\ 0 &\leq 0 \\ 0 &\leq 1 \end{aligned}$$

$Z_{max} = -\ (1,1,0,1)$

$$\begin{aligned} \text{Sbj } 2 &\leq 1 \\ 1 &\leq 1 \\ 0 &\leq 0 \\ 1 &\leq 1 \end{aligned}$$



Optimisation

Analysing the solution found, X1=1 , X2=1 , X3=0 e X4 = 0:

(1,1,0,0)

Max 14

Sbj 0 ≤ 1
0 ≤ 1
0 ≤ 0
0 ≤ 1

We notice that

- It is a solution to the original problem (all variables take integer 0/1 values).
- Its optimum is better than the potential of any of the subproblems not yet exploited (e.g. X1=1,X2=0 **with maximum 13.8)**.
- These may then be safely discarded.

Optimisation

These techniques are the basis of the branch & bound algorithm that is the algorithm usually adopted in a pure constructive approach to optimisation.

The algorithm exploits the following ideas

- A problem may be divided in two (or more) problems, with exclusively disjunct solutions (Branch).
- The potential of each subproblem is evaluated by some appropriate (relaxation) technique (Bound).
- The subproblems that do not have a better potential than the best solution found so far, are abandoned.

Optimisation

During this process, the algorithm has to exploit a (potentially large) number of problems.

In the worst case, problems resulting from all possible instantiations of the n variables would have to be examined, a search space of order $O(d^n)$.

A key issue that must then be considered, to make the algorithm efficient, is the computation as exact as possible of the following bounds, for the unexplored problems

- Their potential (upper bounds in Max problems)
- A guaranteed value of a solution of the problem (lower bounds), enabling cuts to be made even before a solution is found.

Optimisation

Other techniques may improve the algorithm, namely

- Fixing of variables

$$x_3 \leq 0 \quad \Rightarrow \quad x_3 = 0$$

- Elimination of useless redundant constraints

$$x_4 \leq 1$$

- Generation of useful redundant constraints (cutting-planes) .
- Example: From $6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10$ "infer"

$$x_1 + x_3 \leq 1$$

$$x_1 + x_2 + x_4 \leq 2$$

Some of these techniques are automatically guaranteed by constraint propagation

Optimisation

These techniques may be used in finite domains constraints (integer programming), with the obvious adaptations

- Decomposition is often carried out at some threshold value, usually dividing the domain in two equal "parts".
  - If `X::[0..8]` then `X ≤ 4 or X > 4`
  - If `X::[0,1,2,3,4,8]` then `X ≤ 2 or X > 2`
- Useful cutting planes are much more difficult to find.
- Overcome problems with interval arithmetic to compute bounds

Optimisation

The bounds are computed with interval arithmetic, which raises many problems regarding precision, namely if the "same" variables are considered independently.

Ex 1:
```
| ?- X in -2..2, Z #= X*X.
     X in -2..2, Z in 0..4 ?
```

However, the dependencies between the variables are not always recognised by the system.

Ex 2:
```
| ?- X in -2..2, Y #= X, Z #= X*Y.
     X in -2..2, Y in -2..2, Z in -4..4 ?
```

Even worse (!)

Ex 3:
```
| ?- X in -2..2, Y #= X, Z #= X-Y.
     X in -2..2, Y in -2..2, Z in -4..4 ?
```

Optimisation

Expressions on the same variables, equivalent in "normal" arithmetic, may even produce different results.

Ex 1:

```
| ?- X in -2..2, Y in -2..2, Z #= X*(X+Y).
X in -2..2,
Y in -2..2,
Z in -8..8.
```

Ex 2:

```
| ?- X in -2..2, Y in -2..2, Z #= X*X+X*Y.
X in -2..2,
Y in -2..2,
Z in -4..8.
```

Optimisation

Other Topics

Hybrid Solvers CPLEX+CLP (ECLiPSe)

Local Search

Hybridization: Constructive + Repairing

- Construcion of "good" solutions
- Repairing these solutions (improve bound!)

Both Satisfaction and Optimization

Optimisation in SICStus

- *clpfd* library provides **minimize(?Goal, ?C)** (and maximize/2), which finds the solution of Goal that minimizes (maximizes) C (with branch & bound).

```
?- domain([X,Y], 0,6), 2*X+Y #>= 9, Cost#=3*X+2*Y,
   minimize(labeling([],[X,Y]),Cost).
   X = 4,   Y = 1,  Cost = 14
```

- A minimize(Cost) option could be used in labeling/2 instead. E.g. **labeling([minimize(Cost)], [X,Y])**.

- It can also be used in conjunction with time_out/2 option to obtain best solution found in some time limit. E.g. **labeling([maximize(X),time_out(9999,Flag)], Vars)**