Constraint Programming (Modeling)

Course Notes

Nadjib Lazaar
Université Paris-Saclay
lazaar@lisn.fr

Introduction

Constraint programming (CP) is a declarative approach used to model and solve complex combinatorial problems. By defining a set of variables, domains, and constraints, this approach facilitates the expression and automatic resolution of problems without explicitly specifying a particular algorithm.

1 History and Motivation

Constraint programming originated in the 1960s-1970s, particularly in artificial intelligence, logic programming, and operations research. During this time, researchers were exploring different approaches to solving combinatorial problems more efficiently than traditional exhaustive search methods. The fundamental idea was to leverage constraints to reduce the search space and speed up problem-solving.

In the 1980s, the first languages dedicated to constraint programming emerged, notably **CLP (Constraint Logic Programming)**, which combined the declarative logic of Prolog with constraint management. This advancement allowed for better structuring and automation in solving many problems in artificial intelligence and optimization.

In the 1990s-2000s, CP underwent progressive industrialization with the appearance of generic solvers such as *ILOG Solver*, *ECLiPSe*, and *Choco*. These tools facilitated the adoption of CP in various fields such as planning, logistics, and resource optimization.

Today, CP is widely used in industry thanks to high-performance solvers such as *Gecode*, *Google OR-Tools*, and *IBM Ilog CP Optimizer*. These solvers allow for declarative modeling and the use of advanced constraint propagation techniques and heuristic search to quickly find optimal solutions. CP continues

to evolve, particularly with its integration into hybrid approaches combining CP and machine learning.

2 Comparison of Programming Paradigms

CP belongs to the declarative paradigm, which differs from other programming paradigms.

Declarative programming, and particularly CP, stands out for its ability to express problems in terms of logical relationships and constraints, leaving the solver to determine the best possible solution. This approach is particularly useful for combinatorial and optimization problems where traditional methods would be too computationally expensive.

Paradigm	Description	Examples of Languages
Imperative	Describes how to solve a problem using successive instructions	C, Java, Python (imperative mode)
Object-Oriented	Structures the code into objects and their interactions	Java, C++, Python (object mode)
Functional	Based on the evaluation of functions without mutable state	Haskell, Lisp, Scala
Declarative	Specifies the problem to be solved without detailing how to solve it	Prolog, SQL, MiniZinc

Table 1: Comparison of Programming Paradigms

3 Foundations of CP

A constraint satisfaction problem (CSP) is defined by a **finite set of variables**, each taking values from a **finite domain**, and a **set of constraints** expressing relationships between these variables.

- **Variables**: A CSP is defined by a finite set of variables $X = \{x_1, x_2, ..., x_n\}$.
- **Domains**: Each variable x_i has a finite domain $D(x_i)$ containing the values it can take.
- **Constraints**: Constraints are relationships between variables that restrict the possible combinations of values.

3.1 Types of Constraints

Constraints can be defined in various ways:

- **Unary constraints**: These involve a single variable, such as $x_1 \neq 3$.
- **Binary constraints**: These link two variables, such as $x_1 \neq x_2$.
- **k-ary constraints**: These involve more than two variables, such as $x_1 + x_2 + x_3 \leq 10$.

3.2 Constraint Definition Modes

Constraints can be specified in several ways:

- **Extensionally**: By explicitly listing the allowed combinations. For example, for $x_1, x_2 \in \{1, 2, 3\}$:

  $$C = \{(1, 2), (2, 3), (3, 1)\}$$

  meaning that only these pairs of values are allowed for x_1 and x_2.

- **Intentionally**: By defining a logical or mathematical relationship. For example:

  $$x_1 + x_2 = x_3$$

  imposes that the sum of x_1 and x_2 is equal to x_3.

- **Global constraints**: These involve many variables and facilitate constraint propagation. For example:

  $$\texttt{allDifferent}(x_1, x_2, ..., x_n)$$

  requires that all variables have distinct values.

CSP Example

Consider a map coloring problem where we need to color three regions A, B, and C with three different colors (Red, Green, Blue), subject to the following constraint:

- Two adjacent regions cannot have the same color.

CSP formulation:

- **Variables**: $X = \{A, B, C\}$
- **Domains**: $D(A) = D(B) = D(C) = \{Red, Green, Blue\}$
- **Constraints**: $A \neq B$, $B \neq C$, $A \neq C$

The solution involves assigning a color to each region while respecting the defined constraints.

Each constraint restricts the solution space, and the final solution of the CSP is an assignment of variables that satisfies **all** the constraints simultaneously.

4 CP Modeling: Modeling Strategies

Modeling is a crucial step in constraint programming (CP) because it determines the efficiency of problem resolution. A good model relies on several strategies:

4.1 Identification of Variables and Domains

The first step is to define the variables and their domains:

- **What are the key elements of the solution?** These are the entities of the problem that we need to represent as variables.
- **What values can these elements take?** Each variable must be associated with a finite domain of possible values.

4.2 Defining Constraints

Once the variables are defined, the next step is to establish the relationships between them in the form of constraints:

- **Which relationships exist between variables?** These relationships express the restrictions we want to impose on the problem.
- **What are the possible ways of defining constraints?** Constraints can be expressed intensionally (using logical conditions) or extensionally (listing allowed combinations).

4.3 Choice of Granularity Level

It is important to adapt the modeling based on the problem:

- **Does a variable represent a subproblem or an individual value?** Sometimes, a single variable can represent a set of elements.
- **Should certain variables be grouped to simplify the model?** A good structuring of the variables can improve the efficiency of the solution process.

4.4 Modeling Optimization

An effective modeling approach aims to reduce the complexity of the problem:

- **Avoid redundancies.** Do not define unnecessary constraints that would complicate the model.
- **Reduce complexity by limiting the number of constraints.** Simplifying the model improves the solution search speed.
- **Consider constraint propagation to guide the search.** Good propagation helps eliminate inconsistent values quickly and speeds up CSP resolution.

5 Search Space in CSP

The search space of a constraint satisfaction problem (CSP) represents the set of all possible assignments of the problem's variables. It plays a crucial role in the efficiency of the resolution and can be optimized using various techniques.

5.1 Definition of the Search Space

The search space is defined as the Cartesian product of the variable domains:

$$E = D(x_1) \times D(x_2) \times ... \times D(x_n)$$

Thus, if each variable has a domain of d values and there are n variables, then the size of the search space is d^n.

5.2 Reducing the Search Space

The raw search space can be immense, but it is reduced by constraints that filter out invalid assignments. Several techniques help optimize this reduction:

- **Constraint propagation**: eliminating impossible values before exploring the search space.

- **Heuristics for variable and value selection**: strategies to efficiently explore the search space.
- **Problem decomposition**: breaking the problem into smaller subproblems to restrict the search space.
- **Local search**: partial exploration of the space to quickly find approximate solutions.

5.3 Impact of Constraints on the Search Space

Constraints directly influence the size and structure of the search space:

- **Strict constraints**: significantly reduce the search space, making the resolution easier but risking eliminating viable solutions.
- **Soft constraints**: leave more options open but make the search more complex.
- **Global constraints**: allow expressing complex relationships in a single constraint and improve resolution efficiency.

Proper management of the search space is essential to efficiently solve a CSP, avoiding exhaustive exploration and favoring intelligent search techniques.

6 Symmetry Breaking in CSP

Symmetry breaking is an essential technique in CSP that reduces the search space by eliminating equivalent solutions. A symmetry occurs when there are multiple solutions that are identical except for a simple rearrangement of values.

6.1 Why Break Symmetries?

Symmetric problems have multiple equivalent solutions, which can slow down the search by unnecessarily exploring redundant configurations. By adding additional constraints to eliminate these duplications, we reduce computation time and improve solver efficiency.

6.2 Symmetry Breaking Methods

Various approaches exist to manage symmetries:

- **Adding symmetry-breaking constraints**: imposing restrictions to force a unique representation among symmetric solutions. Example: fixing a pivot variable ($x_1 < x_2$) to avoid unnecessary permutations.
- **Reducing the search space**: adjusting search heuristics to avoid re-exploring the same configurations.
- **Dynamic filtering**: removing symmetric solutions during the search using specialized algorithms.

6.3 Example of Symmetry Breaking

Consider the n-queens problem, where we need to place n queens on an $n \times n$ chessboard such that no two queens attack each other. This problem has multiple symmetric solutions due to rotation or reflection. We can break these symmetries by imposing that the first queen is placed in the left half of the chessboard:

Symmetry Breaking Example

n-Queens Problem: Symmetry-breaking constraints applied:

- Fix the first queen on the first row.
- Require that the column of the first queen be less than $n/2$.
- Eliminate solutions obtained by rotation or reflection.

Applying these techniques reduces the number of configurations explored and accelerates the resolution of the CSP.

7 Redundant Constraints in CSP

Redundant constraints are constraints added to a CSP that do not change the set of valid solutions but help accelerate the resolution by enhancing propagation.

7.1 Why Add Redundant Constraints?

Although a redundant constraint does not alter the set of solutions, it can:

- Improve constraint propagation and reduce the search space.
- Eliminate inconsistent values more quickly and avoid unnecessary explorations.
- Facilitate convergence to a solution by providing additional information to the solver.

7.2 Examples of Redundant Constraints

- **Sum of variables**: In a problem where x_1, x_2, x_3 are required to take different values ($\texttt{allDifferent}(x_1, x_2, x_3)$), a redundant constraint like $x_1 + x_2 + x_3 = 6$ can improve propagation.
- **Implicit symmetry**: In an object placement problem on a grid, a constraint requiring the sum of the object indices to be even can speed up the search without changing the set of solutions.

- **Propagation optimization**: In a scheduling problem, adding additional precedence constraints can limit the assignments explored by the solver.

The addition of redundant constraints must be done with caution, as too many constraints can slow down the resolution instead of speeding it up. A good problem analysis helps identify the most relevant constraints to add.

8 Channeling Constraints

Channeling constraints allow the introduction of auxiliary variables to simplify the modeling of a problem by establishing a correspondence between different representations of the same concept.

8.1 Why Use Channeling Constraints?

Adding auxiliary variables and channeling constraints is useful to:

- Facilitate the expression of certain complex constraints.
- Improve constraint propagation and accelerate the search for solutions.
- Provide an equivalent alternative that simplifies the problem resolution.

8.2 Types of Channeling Constraints

Channeling constraints are often used to link different formulations of a problem:

- **Correspondence between boolean and integer variables**: A boolean variable b_i can be introduced to indicate whether an integer variable x_i takes a given value.
$$b_i = 1 \iff x_i = v$$
- **Alternative representation of a constraint**: Two sets of variables can be defined to represent the same information in different ways, with channeling constraints ensuring their consistency.
- **Decomposition of a global constraint**: A complex constraint can be transformed into several local constraints linked by auxiliary variables.

Using channeling constraints is a powerful technique for structuring and simplifying a constraint satisfaction problem while optimizing its resolution.

9 Global Constraints in Modeling

Global constraints are predefined constraints that capture frequent structures in CSPs. They allow for efficient modeling of complex relationships without the need to enumerate all equivalent elementary constraints.

9.1 Why Use Global Constraints?

Global constraints offer several advantages in modeling:

- They allow for expressing a problem in a more compact and readable way.
- They improve constraint propagation, thus reducing the search space.
- They avoid the need to explicitly formulate sets of equivalent constraints.

9.2 Key Global Constraints for Modeling

Here are some commonly used global constraints in CSP modeling:

- `allDifferent`$(x_1, x_2, ..., x_n)$: imposes that all variables take distinct values. This constraint is useful in scheduling and planning problems.
- `sum`$(x_1 + x_2 + ... + x_n = S)$: imposes a given sum over a set of variables. It is frequently used in assignment and optimization problems.
- `element`$(i, [v_1, v_2, ..., v_n], x)$: imposes that the variable x takes the value at position i in a list. This constraint is useful for modeling choices dependent on an index.
- `circuit`$(x_1, x_2, ..., x_n)$: imposes that a set of variables defines a Hamiltonian circuit, commonly used in routing problems.
- `globalCardinality`$(x_1, x_2, ..., x_n, [a_1, ..., a_m], [b_1, ..., b_m])$: restricts the frequency of occurrence of values in a set of variables. It is often used in balanced distribution problems.

Global constraints are powerful tools that make modeling more intuitive and improve the performance of solvers in CSPs.