Stanford MS&E/CS 331: Constraint satisfaction programming

Ellen Vitercik*

October 2, 2025

In this lecture, we continue our effort to establish a common language for discussing optimization problems, with a particular focus on the discrete setting. In the previous class, we introduced integer linear programs, which take the following general form:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ & x_j \in \mathbb{Z} \quad \text{for some or all } j \in [n]. \end{array}$$

While ILPs capture a broad range of problems through linear inequalities and an explicit objective function, constraint satisfaction programming (CSP) provides a more natural framework for many classic combinatorial problems such as graph coloring, sudoku, and 3SAT. CSPs are best suited for problems where constraints are expressed as logical or structural relations. Table 1 summarizes some of the key distinctions between these two modeling frameworks.

CSPs play a central role in many real-world applications. For example, NASA has developed and deployed the open-source EUROPA system, a planning and scheduling framework that has been used across multiple space mission applications. In another domain, chip design verification relies heavily on SAT formulations. Hardware verification problems can often be reduced to these satisfiability problems, allowing SAT solvers to automatically check the correctness of complex circuit designs. These solvers can detect design flaws before a chip is fabricated.

1 General form of a CSP

As a running example in this section, we will use the (NP-hard) max-cut problem. Given a graph $G = (V, E)$, a *cut* in a graph is a subset of its vertices $S \subseteq V$. The *weight* of a cut $w(S)$ is the number of edges that cross S to $V \setminus S$. The goal of this problem is to find a cut with maximum weight.

A CSP can be described formally as an instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$. The set $\mathcal{X}$ contains the variables of the problem. For example, in the max-cut problem we introduce a variable $X_v \in \mathcal{X}$ for every vertex $v \in V$.

Each variable $X \in \mathcal{X}$ is associated with a *domain* of possible values, specified by the mapping $\mathcal{D}(X)$. For example, in max-cut, the domain is binary: $\mathcal{D}(X_v) = 0, 1$, where the

*These notes are course material and have not undergone formal peer review. Please feel free to send me any typos or comments.

	Integer Programming	**CSP**
Variables	Integers (often real-valued bounds)	Finite domain elements
Constraints	Linear inequalities/equations	Arbitrary relations over tuples
Objective	Explicit linear function	Maximize satisfied constraints
Natural lens	Numeric optimization	Logical/structural feasibility

Table 1: Integer programming versus constraint satisfaction programming.

value indicates which side of the cut vertex v belongs to. An *assignment* α then chooses a value $\alpha(X)$ from $\mathcal{D}(X)$ for every variable $X \in \mathcal{X}$.

The third component, $\mathcal{C}$, is the set of *constraints*. Each constraint $C \in \mathcal{C}$ is defined by a *scope*, which is a tuple of variables $s^C = (X_1, \ldots, X_k)$, together with a *relation*, which is a set $R^C \subseteq \mathcal{D}(X_1) \times \cdots \times \mathcal{D}(X_k)$ describing the allowable joint assignments to those variables. In the max-cut example, for each edge $(u, v) \in E$, we add a constraint with scope (X_u, X_v) and relation $\{(0, 1), (1, 0)\}$. This constraint is satisfied precisely when u and v are assigned to opposite sides of the cut.

The overall goal of a CSP is to find an assignment that satisfies as many constraints as possible, ideally all of them.

2 Example: 3SAT

A classic example of a constraint satisfaction problem is 3SAT. The problem is defined over Boolean variables $(X_1, \ldots, X_n)$. A *literal* is either a variable X_i itself or its negation $\neg X_i$, and we denote literals by ℓ_i. A *clause* is the disjunction (logical OR) of three literals, such as $X_1 \vee X_2 \vee \neg X_3$. The goal of 3SAT is to find an assignment of truth values to the variables that maximizes the number of satisfied clauses, or, in the decision version, to determine whether all clauses can be satisfied simultaneously.

We can express 3SAT naturally in the CSP framework. Each variable X_i has the domain $\mathscr{D}(X_i) = \{0, 1\}$, where we interpret $1 = \text{True}$ and $0 = \text{False}$. For each clause $(\ell_i \vee \ell_j \vee \ell_k)$, we define a constraint C with scope $s^C = (X_i, X_j, X_k)$. The associated relation is

$$R^C = \{(u, v, w) \in 0, 1^3 \mid \text{eval}(\ell_i, u) \vee \text{eval}(\ell_j, v) \vee \text{eval}(\ell_k, w) = 1\},$$

where the evaluation function eval enforces the semantics of literals:

$$\text{eval}(X, 1) = 1, \quad \text{eval}(X, 0) = 0, \quad \text{eval}(\neg X, 1) = 0, \quad \text{eval}(\neg X, 0) = 1.$$

An assignment α satisfies a constraint C precisely when the tuple $(\alpha(X_i), \alpha(X_j), \alpha(X_k))$ lies in R^C. For example, consider the clause $X_i \vee \neg X_j \vee X_k)$. The only falsifying assignment is $(0, 1, 0)$, so the relation is $R^C = \{0, 1\}^3 \setminus \{(0, 1, 0)\}$.