PhD-FSTM-2026-025

The Faculty of Science, Technology and Medicine

DISSERTATION

Defence held on 17/03/2026 in Esch-sur-Alzette
to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG EN INFORMATIQUE

by

Hedieh Haddad
Born on 18th March 1994 in Zabol, Iran

HYPERPARAMETER OPTIMIZATION OF CONSTRAINT PROGRAMMING SOLVERS

Dissertation defence committee

Dr. Pascal Bouvry, dissertation supervisor
Professor, University of Luxembourg

Dr. Grégoire Danoy, Chairman
Assistant Professor, University of Luxembourg

Dr. Luis A. Leiva
Associate Professor, University of Luxembourg

Dr. El-Ghazali Talbi
Professor, University of Lille

Dr. Kittichai Lavangnananda
Professor, King Mongkut's University of Technology Thonburi (KMUTT)

Affidavit / Statement of originality
I declare that this thesis:

- is the result of my own work. Any contribution from any other party, and any use of generative artificial intelligence technologies have been duly cited and acknowledged;
- is not substantially the same as any other that I have submitted, and;
- is not being concurrently submitted for a degree, diploma or other qualification at the University of Luxembourg or any other University or similar institution except as specified in the text.

With my signature I furthermore confirm the following:

- I have adhered to the rules set out in the University of Luxembourg's Code of Conduct and the Doctoral Education Agreement, in particular with regard to research integrity.
- I have documented all methods, data, and processes truthfully and fully.
- I have mentioned all the significant contributors to the work.
- I am aware that the work may be screened electronically for originality.

I acknowledge that if any issues are raised regarding good research practices based on the review of the thesis, the examination may be postponed pending the outcome of any investigation of such issues. If a degree was conferred, any such subsequently discovered issues may result in the cancellation of the degree.

In Esch-sur-Alzette date 2026-01-16 

Author's signature

A PhD Dissertation by

Hedieh Haddad

Submitted to the University of Luxembourg

HYPERPARAMETER OPTIMIZATION OF CONSTRAINT PROGRAMMING SOLVERS

Members of the thesis supervision committee (CET):

Supervisor:	Prof. Dr. Pascal BOUVRY, University of Luxembourg
Committee members:	Prof. Dr. Grégoire Danoy, University of Luxembourg Prof. Dr. Luis LEIVA, University of Luxembourg

To my husband—my dearest friend, Javad, who has always believed in me, stood by my side throughout this journey, and whose constant support and encouragement made it possible to complete what I had begun.

To my family, who have accompanied me on this journey with unwavering love, patience, and faith.

And to all who seek knowledge and understanding, may this work serve, in some small way, as a contribution to our shared pursuit of truth.

Abstract

The performance of constraint programming solvers is highly sensitive to the configuration of search heuristics, restart policies, and propagation levels. Manually tuning these hyperparameters is inefficient, expertise-dependent, and rarely generalizes across diverse problem instances. This thesis addresses this fundamental bottleneck by automating solver configuration through formalized hyperparameter optimization, aiming to create adaptive, reproducible, and self-configuring constraint solvers.

In the first part, we establish the theoretical and practical foundations. We formalize solver configuration as a hyperparameter optimization problem over discrete, categorical, and often conditional parameter spaces. We then conduct a comprehensive evaluation of optimization strategies, including grid search, random search, Hamming-distance local search, Bayesian optimization, and Hyperband, within the context of constraint programming. This analysis reveals that model-based methods, particularly Bayesian optimization, offer superior sample efficiency and robustness under strict time budgets, providing a principled basis for automated configuration.

Building on these insights, the core contribution of this thesis is the introduction of the probe and solve algorithm, a novel, solver-agnostic framework for time-budgeted hyperparameter optimization. The framework operates in two adaptive phases: a probing phase for exploring the configuration space and a solving phase that executes with the most promising configuration. It incorporates key mechanisms such as dynamic timeout scheduling, stagnation-aware early stopping, and memory-based caching to maximize efficiency without requiring internal solver modifications. This design enables robust, near-parameterless operation across varying computational constraints.

Next, we present an extensive empirical validation of the framework. Through large-scale benchmarking on the ACE and Choco solvers using XCSP3 and MiniZinc problem sets, we demonstrate that the probe and solve algorithm consistently outperforms default solver configurations. We perform a full-factorial analysis of its modular components, time allocation, timeout evolution, and stopping conditions, to identify optimal configurations and validate the framework's generalization across solver architectures and modeling environments.

In the final part, we transition from solver-centric optimization to real-world application within the architecture, engineering, and construction domain. We develop an integrated pipeline that transforms building information models and textual engineering standards, such as ISO 10077-1 for thermal performance and ISO 717-1 for acoustic insulation, into executable constraint programming models. This enables automated, multi-objective design optimization with guaranteed regulatory compliance. We further enhance this pipeline with a hierarchical search strategy, combining the probe and solve algorithm for learned intensification with large neighbourhood search for structural exploration, resulting in a comprehensive method for generating diverse, high-quality Pareto-optimal design solutions.

This thesis bridges algorithmic innovation with practical application, advancing both automated solver configuration and standards-aware optimization in combinatorial problem solving.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Dr. Pascal Bouvry, for granting me the opportunity to pursue this research. This PhD would not have been possible without his trust, and constant support. His vision and encouragement have shaped my scientific journey and provided me with the freedom to explore and grow as a researcher.

My sincere appreciation goes to Dr. Pierre Talbot, my primary scientific advisor, whose exceptional expertise, patience, and dedication have been fundamental to every stage of this work. I could not have wished for a better advisor, one who has been continuously by my side, guiding me with insight, clarity, and generosity. His deep knowledge and sharp intuition have inspired me to pursue research with rigor and curiosity.

I would also like to extend my heartfelt thanks to Prof. Grégoire Danoy, head of the Parallel Computing and Optimization Group (PCOG), for his leadership and for fostering such an inspiring and collaborative research environment. It has been a privilege to work in a group marked by respect, openness, and genuine scientific passion.

I would also like to thank my colleagues, Ms. Maria Hartmann, Mr. Manuel Combarro Simon, and Ms. Jingjing Xu, for their collaboration, and encouragement. They were not only colleagues but true friends, whose support, laughter, and companionship made this journey both rewarding and enjoyable.

I would like to express my deepest gratitude to my wonderful family for their unwavering love, support, and encouragement throughout this journey. A very special thanks to my beloved husband, whose love, kindness, and understanding have been my greatest sources of strength. His constant encouragement and belief in me have carried me through every challenge, and his presence has made this journey truly meaningful.

Among all, my deepest gratitude and love go to my mother. She watched me take my first steps on this journey and supported me with every ounce of her energy and heart, but she could not be here to see me reach its end. I know she is watching from above, and I dedicate this achievement to her memory—with endless love and gratitude.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AEC Architecture, Engineering, and Construction.
AI Artificial Intelligence.

BIM Building Information Modelling.
BO Bayesian Optimization.

COP Constraint Optimization Problem.
CP Constraint Programming.
CPU Central Processing Unit.
CSP Constraint Satisfaction Problem.

DT Decision Tree.

FNR Fonds National de la Recherche.

GP Gaussian Process.
GT Global Timeout.

HP Hyperparameter.
HPO Hyperparameter Optimization.
HV Hypervolume.

IFC Industry Foundation Classes.
ILNAS Institut Luxembourgeois de la Normalisation, de l'Accréditation, de la Sécurité et qualité des produits et services.
ISO International Organization for Standardization.

LNS Large Neighbourhood Search.

MIP Mixed Integer Programming.
ML Machine Learning.
MOO Multi-Objective Optimization.

NLP Natural Language Processing.
NSGA-II Non-dominated Sorting Genetic Algorithm II.

OR Operations Research.

PSA Probe-and-Solve Algorithm.

RTD Run-Time Distribution.

$\mathbf{S_{val}}$ Value Selection Heuristic.
$\mathbf{S_{var}}$ Variable Selection Heuristic.

SAT Boolean Satisfiability Problem.

TG Global Timeout.
TP Probing Timeout.
TS Solving Timeout.

Uw Thermal Transmittance of the Window.

XCSP3 XML Constraint Satisfaction Problem Format (version 3).

Chapter 1

Introduction

Contents

1.1 Context

Constraint programming (CP) is a powerful paradigm for solving combinatorial problems that are defined by a set of variables and constraints that specify the relationships among them. It provides a declarative modeling approach where the user specifies what needs to be satisfied rather than how to achieve it. Over the past decades, CP has proven effective across a wide range of domains, such as scheduling, planning, configuration, and design [89, 3].

The performance of CP solvers largely depends on how effectively they explore the search space and how their parameters are configured. Despite their general-purpose design, solver efficiency can vary significantly depending on the chosen search strategy and parameter values. To address such variability in search behavior and the wide range of possible solver configurations, other fields like machine learning (ML) have successfully employed hyperparameter optimization (HPO) to automate the selection of effective parameter settings [35, 37]. However, while HPO methods have been applied to CP solvers [16], existing approaches often require solver-specific adaptations or extensive meta-parameter tuning themselves. This thesis

addresses these limitations by developing a general, solver-agnostic framework that requires minimal user intervention.

1.2 Motivation

Designing efficient search strategies in CP is a challenging process that typically relies on expert intuition, significant domain knowledge, and extensive manual experimentation. Despite decades of research, no single search heuristic or solver configuration consistently performs well across diverse problem structures [95, 100]. As a result, identifying effective solver settings remains problem-dependent and time-consuming.

Modern CP solvers expose a growing number of configurable parameters, such as variable ordering heuristics, value selection rules, restart policies, and propagation strengths [89]. These hyperparameters can have a substantial impact on performance, but systematically tuning them is difficult. Manual tuning not only demands expertise but can fail to fully exploit the solver's performance potential.

In parallel, advances in automated algorithm configuration, especially HPO techniques, have demonstrated strong results in ML and optimization [53]. Model-based approaches like Bayesian optimization (BO) are particularly appealing, as they construct surrogate models of performance and leverage them to explore large configuration spaces efficiently [94]. This enables intelligent decision-making about which configurations to evaluate next, outperforming exhaustive or uninformed search heuristics such as grid search or random search [11].

These developments create a compelling opportunity: by treating solver search strategies and controls as hyperparameters, HPO can be used to automatically identify configurations that deliver strong performance on new and diverse CP problems. This reduces the burden on users, limits manual trial-and-error, and contributes toward making CP more accessible, scalable, and widely applicable.

Furthermore, in applied domains such as architecture, engineering, and construction (AEC), engineering standards (e.g., thermal, acoustic regulations) are typically expressed textually, requiring manual interpretation. Automating compliance checking and optimization in such domains demands not only effective solver configuration but also the ability to encode normative knowledge as computable constraints—a task well-suited to CP once the configuration challenge is addressed.

1.3 Problem Statement and Research Questions

1.3.1 Problem Statement

The central problem addressed in this thesis is the lack of general and automated methods for selecting optimal hyperparameter configurations in CP solvers. Manual configuration is both time-consuming and inefficient, while exhaustive search methods such as grid search are computationally expensive. A critical challenge in automated algorithm configuration is avoiding the introduction of new hyperparameters that require manual tuning. If an HPO algorithm itself needs extensive parameter configuration, it merely shifts rather than solves the configuration problem. Therefore, there is a need for intelligent, data-driven methods that can learn and adapt solver configurations efficiently while operating in a parameterless or near-parameterless manner.

This challenge extends to real-world applications where CP is used to model regulated systems, such as building design under engineering standards. An effective HPO method should therefore also support such applied scenarios, enabling automated, standards-aware optimization without requiring additional manual tuning.

The primary objective of this thesis is to develop, implement, and evaluate an automated framework for HPO in CP solvers that operates with minimal user-defined parameters. This framework aims to enhance solver performance by leveraging adaptive and learning-based techniques, ultimately reducing the gap between theoretical solver capabilities and their practical performance on diverse problem instances.

1.3.2 Research Questions

To achieve this objective, the thesis seeks to answer the following research questions:

- **RQ1:** Can HPO methods be effectively applied to the discrete and complex hyperparameter spaces of CP solvers, and how can they be used to operate efficiently in this context?
- **RQ2:** Among different HPO paradigms, model-free (e.g., random search, Hyperband) and model-based (e.g., BO), which approaches are best suited for configuring CP solvers under realistic time constraints?
- **RQ3:** How can the approach be made fully parameterless while maintaining robustness across different allocated times?

- **RQ4:** To what extent does this approach generalize across solvers and modeling environments, and how consistent is its performance under varying computational budgets?
- **RQ5:** How can solver performance be systematically benchmarked across different HPO strategies to identify their strengths and limitations?
- **RQ6:** Can an integrated HPO framework be designed to generalize across different solvers, problem types, and domains?

1.4 Overview of the Probe and Solve Algorithm

To address these questions, we introduce the **probe and solve algorithm (PSA)**—a simple, non-invasive, two-phase HPO framework for CP solvers. PSA divides the total solving time into:

1. **Probing phase:** A short initial period (typically 20% of the total time) where different solver configurations are evaluated using an HPO method (e.g., BO). Each evaluation uses adaptive timeouts to quickly assess configuration quality, with mechanisms like stagnation detection to avoid wasted exploration.
2. **Solving phase:** The remaining time is dedicated to solving the problem with the best configuration found during probing, optionally enforced with objective cut constraints to ensure improvement over the probing phase.

PSA requires no internal solver modifications, works with any HPO backend, and automatically adapts to problem difficulty through different mechanisms; this modular design is detailed in Chapter 3. This approach balances *exploration* (finding good settings) with *exploitation* (using them effectively), operating in a nearly parameterless, solver-agnostic manner.

1.5 Contributions

The key contributions of this thesis are summarized as follows, with forward references to the chapters and sections where they are developed and validated:

1. **Framework Design (Chapter 3):** Development of the *PSA*, a unified hyperparameter optimization framework that addresses the configuration challenge through a simple yet effective two-phase approach.

2. **Algorithmic Integration and HPO Comparison (Section 4.2):** Implementation and comparison of multiple HPO strategies within PSA—including BO, random search, and Hyperband—enabling both model-based and model-free optimization. Experimental results show that BO with a 20% probing ratio yields the best performance across diverse benchmarks.

3. **Solver Integration and Generalization (Section 4.2):** Extension of the CP modeling library `CPMpy` [70] to support new solver interfaces, including the integration of the ACE solver via PyCSP3 [68], facilitating advanced experimentation across different solving backends. PSA demonstrates consistent performance across ACE/XCSP3 and Choco/MiniZinc ecosystems.

4. **Large-Scale Benchmarking and Component Analysis (Section 4.3):** Comprehensive empirical evaluation on complete solver configurations, identifying optimal PSA settings: static round timeout, geometric timeout evolution, and timeout-based stopping. PSA outperforms default solver configurations and simpler tuning methods like Hamming distance search, winning on over 45% of problem instances.

5. **Real-World Application and Standardization (Chapter 5, 6):** Formalization of engineering standards as constraints within CP models, enabling automated, standards-compliant multi-objective optimization (MOO) design in construction workflows. This bridges CP-based optimization with industry practices and regulatory compliance.

1.6 Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2** presents the theoretical foundations of CP, HPO methods, HPO for CP solvers, and MOO.

- **Chapter 3** introduces the complete PSA framework for automated search strategy selection in CP, detailing its two-phase architecture and modular components.

- **Chapter 4** provides a comprehensive empirical evaluation of PSA in two complementary parts:

    1. **PSA on Restricted Hyperparameter Spaces (Section 4.2)** comparing HPO methods and testing generalization across solvers and modeling frameworks.

2. **PSA on Complete Solver Configurations (Section 4.3)** conducting large-scale, full-factorial studies to analyze PSA components and identify champion configurations.

- **Chapter 5** presents preliminary work on integrating construction standards into CP models, demonstrating how standards can be encoded as computable constraints.
- **Chapter 6** demonstrates real-world applications in construction, integrating building information models (BIM) and computable standards for multi-objective design, with case studies on thermal and acoustic performance optimization.
- **Chapter 7** summarizes the contributions, discusses limitations and threats to validity, and outlines future research directions.

Part I

Background & State of the Art

Chapter 2

Background and State of the Art

Contents

This chapter establishes the theoretical foundations and reviews the state of the art relevant to this thesis. We first introduce CP, focusing on its core solving mechanisms and the critical role of search strategies. Subsequently, we examine general HPO methods, detailing

established approaches from simple baselines to advanced model-based techniques. We then focus specifically on the application of HPO to CP solvers, reviewing the current research landscape in automated algorithm configuration for constraint solving. Following this, we present MOO concepts and their relevance to both CP and HPO. Finally, we contextualize these technologies within the domain of AEC, highlighting the challenges of digitalizing engineering standards and the opportunities for computable design automation.

2.1 Constraint Programming

2.1.1 Formal Definition and Notations

CP is a declarative paradigm for solving complex combinatorial problems. Its core principle is the separation of problem modeling from the solving algorithm, allowing users to state the problem's logic without specifying the exact steps to find a solution [69].

At its foundation, a CP model is defined as a constraint satisfaction problem (CSP) [3]. A CSP is a triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where:

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a finite set of variables.
- $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ is a set of domains, where each $D(x_i) \subseteq \mathbb{Z}$ is a finite set of possible values for variable x_i.
- $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a set of constraints that restrict the allowed combinations of values for variables in their scope.

An assignment is a mapping that associates each variable $x_i \in \mathcal{X}$ with one value from its domain $D(x_i)$.

A solution to a CSP is an assignment that satisfies all constraints in $\mathcal{C}$. Let $\mathcal{A}$ denote the set of all possible assignments:

$$\mathcal{A} = \{A \mid A : \mathcal{X} \to \bigcup_{x_i \in \mathcal{X}} D(x_i),\ A(x_i) \in D(x_i) \text{ for all } x_i \in \mathcal{X}\}.$$

The set of *solutions* is then $\mathcal{S} = \{A \in \mathcal{A} \mid A \text{ satisfies all constraints in } \mathcal{C}\}$.

Many real-world problems require finding not just *any* solution, but the *best* one according to some criterion. This extends the CSP into a constraint optimization problem (COP). A COP is formally defined as a quadruplet $(\mathcal{X}, \mathcal{D}, \mathcal{C}, f)$, where $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a CSP and f is an *objective function.* This function maps every assignment of variables to an integer value. Formally, if $\mathcal{A}$

is the set of all possible assignments, the signature is $f : \mathcal{A} \to \mathbb{Z}$. The goal of a COP solver is to find a solution that satisfies all the constraints of $\mathcal{C}$ while minimizing (or maximizing) the value of f [69, 49].

2.1.2 Constraint Modeling and Solving

Modeling maps real problems to decision variables, domains, and constraints. Global constraints (e.g., AllDifferent, Cumulative) capture common structures and enable stronger propagation [89, 3]. Effective models balance expressiveness with solver efficiency, often employing decompositions, symmetry breaking, and problem-specific reformulations to tighten relaxations and shrink the search space [49].

State-of-the-art CP solvers interleave systematic tree search with inference via domain filtering (*propagation*). Node selection determines which subproblem to expand; constraint propagators enforce local consistency (e.g., arc or bounds consistency) to prune values; restarts and nogood learning diversify search and prevent stagnation [3, 49]. Solver performance is highly sensitive to these algorithmic choices and to problem structure.

2.1.3 Search and Propagation

Modern CP solvers typically employ a generic search algorithm based on backtracking, which systematically explores the space of possible assignments. This search is tightly coupled with *constraint propagation*, a form of logical inference that removes values from variable domains that cannot be part of any feasible solution [2, 13]. By actively pruning inconsistent values, propagation significantly reduces the size of the search space that must be explored.

Search strategies

The efficiency of a CP solver is critically dependent on its *search strategy*, which determines the order in which the solution space is explored. A standard strategy consists of two components [28]:

1. **Variable Selection:** Decides which unassigned variable to branch on next. Common heuristics include static orderings (e.g., `input_order`) or dynamic strategies based on the current state of domains, such as `first_fail` (selecting the variable with the smallest remaining domain) or generic state-of-the-art heuristics like `dom/wdeg` [18].

2. **Value Selection:** Decides which value from the selected variable's domain to try first. Examples include trying the minimum value (`indomain_min`), maximum value, or a

random value.

Despite the existence of robust default strategies, no single heuristic guarantees superior performance across all problem types [92]. Consequently, selecting the most effective search strategy for a specific problem instance remains a complex task often requiring expert intuition.

2.1.4 Performance Indicators and Solution Scenarios

Common indicators include (i) solve rate under a time budget, (ii) time-to-solution, (iii) best objective value found or bound at timeout, (iv) number of failures/nodes, (v) proof of satisfiability or unsatisfiability, and (vi) anytime solution quality traces. Aggregated or multi-criteria measures are used to mitigate heavy-tailed run-time noise and instance heterogeneity [78].

We distinguish feasibility-only scenarios (find *any* solution), optimization scenarios (prove optimality or improve bounds), and anytime scenarios where high-quality solutions early are preferred over late optimality. Strategy and configuration selection should align with the scenario, instance features, and time budget.

2.1.5 Illustrative Example: Bounded Knapsack

A classic example of a COP is the bounded knapsack problem [62]. Given a set of items, each with a weight and a value, the goal is to determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. A CP model for this can be formulated as follows:

- **Variables:** A set of N integer variables, $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$, where x_i represents the number of times item i is taken.
- **Domains:** Each variable x_i has the domain $D(x_i) = \{0, 1, \ldots, k_i\}$, where k_i is an upper bound on the quantity of item i.
- **Constraint:** The sum of the weights of the chosen items must not exceed the knapsack's capacity, W_{max}:
$$\sum_{i=1}^{N} w_i \cdot x_i \leq W_{max}$$
- **Objective function:** The goal is to maximize the total value of the items in the knapsack:
$$\text{maximize} \quad f(\mathcal{X}) = \sum_{i=1}^{N} v_i \cdot x_i$$

A solver's task is to find an assignment for $(x_1, \ldots, x_N)$ that satisfies the weight constraint and maximizes the objective function.

2.2 Hyperparameter Optimization

2.2.1 Formal Definition and Notations

HPO is the process of automating the selection of an optimal set of hyperparameters for a given learning algorithm or solver. Formally, we consider an algorithm $\mathcal{S}$ (CP solver) and a hyperparameter space Λ. In general, HPO requires defining an objective (or loss) function L that quantifies the performance of the solver for a given set of hyperparameters $\lambda \in \Lambda$. This performance metric is typically empirical, calculated by running the solver $\mathcal{S}$ with configuration λ on a set of problem instances and measuring, for example, the mean runtime or the quality of the solution found [15]. The objective function for HPO is therefore a black-box function $L : \Lambda \to \mathbb{R}$ [37]. The goal of HPO is to find the configuration λ^* that minimizes this function:

$$\lambda^* = \arg\min_{\lambda \in \Lambda} L(\lambda).$$

It is crucial to distinguish between the COP objective function $f(a)$, which measures the quality of a single problem solution, and the HPO objective function $L(\lambda)$, which measures the performance of a solver configuration.

2.2.2 Hyperparameter Optimization Methods

HPO methods vary in how they explore the space, whether they rely on models, and how they allocate evaluation budgets. Below, we outline representative approaches using a common notation.

2.2.2.1 Grid Search

Grid search is a simple yet exhaustive method that evaluates all possible combinations of hyperparameter values within predefined ranges. Although computationally expensive, it provides a useful baseline because it systematically explores the entire hyperparameter space [72]. Let Λ denote the hyperparameter space, expressed as the Cartesian product of the individual hyperparameter domains:

$$\Lambda = \mathcal{H}_1 \times \mathcal{H}_2 \times \cdots \times \mathcal{H}_k,$$

where each $\mathcal{H}_i$ represents the discrete set of candidate values for the i-th hyperparameter. A configuration $\lambda = (\lambda_1, \ldots, \lambda_k)$ corresponds to one choice of values, where $\lambda_i \in \mathcal{H}_i$. The total number of configurations is therefore:

$$N = \prod_{i=1}^{k} |\mathcal{H}_i|.$$

The objective function $L(\lambda)$ is evaluated for each configuration $\lambda \in \Lambda$, and the configuration that achieves the lowest value is selected as the best one: $\lambda^* = \arg\min_{\lambda \in \Lambda} L(\lambda)$. While grid search guarantees complete coverage, its computational cost grows exponentially with the number of hyperparameters [15].

2.2.2.2 Iterative Search Methods: Random and Local Search

Moving beyond exhaustive methods like grid search, iterative strategies explore the hyperparameter space sequentially. Two fundamental and contrasting approaches are random search and local search, which represent the core principles of global exploration and local exploitation, respectively.

Random search is an uninformed method that focuses purely on exploration. It samples a set of hyperparameters λ at random from the hyperparameter space Λ until a predefined budget is met. The key advantage is its effectiveness in high-dimensional spaces where only a few hyperparameters are critical. Unlike grid search, random search explores a more diverse set of values, increasing the likelihood of finding the optimal configuration [11]. However, its major drawback is that it does not learn from past evaluations; every trial is independent, which can lead to inefficiently resampling unpromising regions [12].

In direct contrast, local search is a method centered on exploitation. It begins with an initial configuration and iteratively moves to an adjacent or neighboring configuration only if it offers improved performance. The concept of a neighborhood is crucial, especially in spaces with categorical hyperparameters [52]. This raises the challenge of how to define a neighbor in a discrete space; the standard approach is to use a distance metric, with the Hamming distance being one of the most fundamental metrics. This is because it is particularly well-suited for categorical data, which has no inherent ordering, by simply counting the number of differing hyperparameter values between two configurations [52]. Its simplicity and efficient computation make it a valuable and straightforward method for defining a neighborhood in these discrete spaces.

2.2.2.3 Hamming Distance

The Hamming distance is a standard metric that measures dissimilarity by counting the positions at which two configuration vectors differ:

$$d_H(A, B) = \sum_{i=1}^{n} \mathbb{I}(A_i \neq B_i)$$

where $\mathbb{I}(\cdot)$ is the indicator function, which evaluates to 1 if its argument is true and 0 otherwise. The neighborhood of a configuration λ in Λ, denoted $\mathcal{N}(\lambda)$, is defined as:

$$\mathcal{N}(\lambda) = \{\lambda' \in \Lambda \mid d_H(\lambda, \lambda') = 1\}.$$

These methods exemplify the classic exploration–exploitation trade-off. Random search wastes evaluations in unpromising regions (inefficient exploitation), whereas pure local search can easily become trapped in local optima (poor exploration).

These challenges have led to hybrid optimization strategies. For instance, the iterated local search (ILS) metaheuristic [53] is a simple but powerful method. ILS works by repeatedly applying a local search to a solution, then changing it (perturbing) to escape local optima. By switching between refining a solution locally and making random changes, ILS can effectively explore new and promising areas.

A good example of this is ParamILS, an algorithm that uses the ILS approach [54]. ParamILS searches locally for the best solution, and when it gets stuck, it makes a random change to start searching in a new area. While these iterative methods are better than simple techniques like grid search, they have limits. This shows we need smarter, model-based approaches that intelligently balance exploration and exploitation.

2.2.2.4 Bayesian Optimization

BO is a powerful, model-based HPO strategy designed to optimize expensive black-box functions. This makes it particularly well suited for tuning computationally costly CP solvers. Unlike uninformed methods, BO builds a probabilistic surrogate model to approximate the objective function.

The method builds a surrogate model $g(x)$ of the true objective function $L(x)$, and selects the next configuration x_{n+1} to evaluate by optimizing an acquisition function $a(x)$. The role of the acquisition function is to balance the trade-off between exploring uncertain regions of the hyperparameter space and exploiting regions known to have good performance [22]. The most common choice for this surrogate is a GP [87], which defines a prior over functions. After

observing some data, this is updated to a posterior distribution that models our belief about the objective function's behavior, providing a mean prediction and an uncertainty estimate for any given configuration:

$$g(x) \sim \mathcal{GP}\Big(\mu(x), k(x, x')\Big).$$

where:

- $\mu(x)$ is the mean function, representing the expected value of $L(x)$.
- $k(x, x')$ is the covariance or kernel function, which measures the similarity between points x and x'. It models the correlation between the function values at those points.

This surrogate model is then used by an acquisition function to intelligently guide the search for the next configuration to evaluate [66, 41]. The role of the acquisition function is to balance the critical exploration (probing regions where the model is highly uncertain, which could potentially hide an even better, undiscovered optimum) and exploitation (focusing on regions that the surrogate model predicts will yield high performance) trade-off [21].

Standard acquisition functions like expected improvement (EI) [61] or upper confidence bound (UCB) [97] quantify this potential and select the configuration that offers the best balance. The BO process is iterative: after each new configuration is evaluated, the result is used to update the surrogate model [94]. This allows the search to become progressively more informed, concentrating its evaluations in the most promising areas of the hyperparameter space. By building this explicit model, BO aims to find high-quality solutions with significantly fewer evaluations, a crucial advantage in the CP domain [38].

2.2.2.5 Hyperband Optimization

Hyperband is a budget-allocation strategy that adaptively distributes resources across configurations, building on *successive halving*: many configurations are tested with small budgets and only the best advance [71].

Let R be the maximum budget per configuration, $r_{\min}$ the minimum budget, and $\eta > 1$ the downsampling factor. Define $s_{\max} = \lfloor \log_\eta(R/r_{\min}) \rfloor$. For each bracket $s \in \{s_{\max}, \dots, 0\}$, initialize

$$n^{(s)} = \left\lceil \frac{s_{\max} + 1}{s + 1}\, \eta^s \right\rceil, \qquad r^{(s)} = R\, \eta^{-s}.$$

Within the bracket, at round $i = 0, \dots, s$:

$$n_i = \left\lfloor n^{(s)}\, \eta^{-i} \right\rfloor, \qquad r_i = r^{(s)}\, \eta^i,$$

evaluate each configuration λ to obtain a budgeted loss $L(\lambda; r_i)$, and keep the top $\lfloor n_i/\eta \rfloor$ (for minimization) for the next round. Since $r_s = R$, the final round evaluates survivors at the maximum budget. Hyperband returns the best configuration observed at budget R:

$$\lambda^* \in \arg \min_{\lambda \text{ evaluated at } R} L(\lambda; R).$$

In CP, r typically corresponds to a time limit or a node/propagation budget. Hyperband performs well under tight tuning budgets by quickly discarding weak configurations and offering strong anytime behavior [71, 96].

2.2.3 Performance Indicators and Solution Scenarios

We evaluate HPO by (i) best loss achieved within a wall-clock or evaluation budget, (ii) time to reach a target loss, (iii) stability across random seeds and instance subsets, and (iv) generalization across held-out instances/solvers. For CP, indicators often include solve rate, time-to-first-solution, best objective at timeout, and optimality gaps aggregated over benchmarks [15, 78].

Scenarios include *per-instance* tuning (optimize λ for a single instance), *per-family* tuning (learn λ robust across a class), and *portfolio* construction (select a set of complementary λ's). Under small budgets typical for CP, model-based BO and budget-aware Hyperband are attractive; random/Hamming search remain competitive baselines [15, 11].

2.2.4 Illustrative Example: Bounded Knapsack

Continuing with the bounded knapsack example, a CP solver must decide which variable to branch on next (e.g., the item with the best value-to-weight ratio) and which value to try first (e.g., the maximum possible quantity). These choices are key hyperparameters that define the search strategy.

- **Hyperparameter space** Λ: The space of all possible search strategies. For instance, a single configuration $\lambda_1 \in \Lambda$ could be a "greedy" strategy:

$$\lambda_1 = \{\text{var\_select: } \texttt{'max\_value\_ratio'}, \text{ val\_select: } \texttt{'indomain\_max'}\}$$

  Another configuration, $\lambda_2 \in \Lambda$, representing a more conservative strategy, could be:

$$\lambda_2 = \{\text{var\_select: } \texttt{'first\_fail'}, \text{ val\_select: } \texttt{'indomain\_min'}\}$$

- **HPO objective function** $L(\lambda)$: To evaluate these strategies on a large knapsack instance, we could define $L(\lambda)$ as the best objective value found by the solver within a fixed time limit when using the strategy defined by λ.

- **HPO goal**: The HPO process would then automatically test different configurations like λ_1, λ_2, and many others to find the one, λ^*, that finds the better total value for the knapsack within the given time.

2.3 Hyperparameter Optimization for Constraint Programming Solvers

The application of HPO to CP solvers represents a specialized subfield within algorithm configuration, driven by the need to automate the selection of effective search strategies and solver parameters. This section reviews the state of the art in this domain, building upon the general HPO methods introduced in Section 2.2.

2.3.1 Automated Algorithm Configuration for CP

The challenge of configuring CP solvers has evolved from manual tuning to systematic automated approaches. Early work in this area established the foundation for what is now called automated algorithm configuration (AAC). The algorithm configuration problem can be formalized as finding the optimal parameter configuration λ^* for a target algorithm A on a set of problem instances $\mathcal{I}$, given a performance metric m:

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} m(A(\lambda, I))$$

where $A(\lambda, I)$ denotes the run of algorithm A with configuration λ on instance I [53].

One of the pioneering systems in this domain was *ParamILS* (iterated local search in parameter space) [54]. ParamILS employs stochastic local search combined with random restarts to explore the parameter space. It was particularly effective for discrete parameter spaces and demonstrated significant improvements over default configurations for various algorithms. Following this, *IRACE* (iterated racing for automatic algorithm configuration) [73] introduced a racing approach that evaluates configurations in parallel and eliminates poorly performing ones early, efficiently allocating computational resources.

The *sequential model-based algorithm configuration (SMAC)* framework represented a signif-

icant advancement by incorporating model-based reasoning [52]. SMAC uses random forests as surrogate models to predict algorithm performance and employs expected improvement as an acquisition function to guide the search. This model-based approach proved particularly effective for mixed continuous-categorical parameter spaces common in algorithm configuration.

2.3.2 Specialized Challenges in CP Solver Configuration

Configuring CP solvers presents unique challenges that distinguish it from other algorithm configuration domains:

1. **Heterogeneous Parameter Spaces**: CP solvers typically expose a large number of parameters controlling search strategies, propagation algorithms, restart policies, and inference levels. These parameters are often hierarchical and conditional, creating complex configuration spaces. For instance, the choice of variable selection heuristic may enable or disable specific parameters controlling that heuristic's behavior [16].

2. **Categorical and Conditional Parameters**: Unlike ML hyperparameters that are often continuous, CP solver parameters are predominantly categorical (e.g., choice of variable selection heuristic: `dom/wdeg`, `first_fail`, `impact`) and conditional (certain parameters only become relevant when specific heuristics are selected) [52].

3. **Performance Variability**: CP solver performance exhibits high variability across problem instances, even within the same problem class. This heavy-tailed runtime distribution necessitates robust configuration methods that consider aggregated performance across diverse instances [42].

4. **Expensive Evaluations**: Evaluating a single configuration can be computationally expensive, as it may require solving multiple problem instances to obtain reliable performance estimates. This makes sample efficiency a critical concern in CP solver configuration [52, 25].

2.3.3 Recent Advances in CP Solver Configuration

Recent research has explored several promising directions for improving CP solver configuration:

2.3.3.1 Bayesian Optimization for CP

BO has emerged as a particularly effective approach for CP solver configuration due to its sample efficiency. For instance, BO has emerged as a particularly effective approach for configuring CP solvers due to its sample efficiency. For instance, model-based configuration can tune solver parameters used as back-end engines in the MiniZinc/FlatZinc toolchain [76, 25]. Similarly, it has been shown effective for configuring the OR-Tools CP-SAT solver [81], outperforming simpler methods like random search for complex configuration spaces [16].

A key innovation in this area has been the development of specialized kernels for categorical parameters. While traditional BO often uses Gaussian processes with standard kernels (e.g., RBF, Matérn) designed for continuous spaces, CP configuration requires handling categorical parameters. Recent work has explored the use of specialized distance metrics and kernels for categorical spaces, such as the Hamming distance kernel for categorical variables and tree-structured Parzen estimators (TPE) that are particularly well-suited for hierarchical parameter spaces [12].

2.3.3.2 Multi-fidelity and Budget-aware Methods

Given the high cost of evaluating CP solver configurations, multi-fidelity methods have gained attention. These approaches use cheaper, approximate evaluations (e.g., with shorter time limits, smaller instances, or reduced search budgets) to filter out poor configurations before committing to expensive full evaluations. Hyperband [71] and its Bayesian extension BOHB [36] have been particularly successful in this context. These methods are especially relevant for CP, where preliminary evaluations with reduced node limits or time budgets can provide meaningful signals about configuration quality [16].

2.3.3.3 Instance-Specific and Adaptive Configuration

Rather than seeking a single optimal configuration for all instances, instance-specific configuration aims to select the best configuration for each problem instance based on its features. This approach recognizes that different problem characteristics may benefit from different solver strategies. Iommazzo et al. [58] proposed a method for predicting the best CP solver configuration based on instance features, demonstrating significant performance improvements over static configurations. Similarly, learning portfolio methods create sets of complementary configurations that can be selected based on instance characteristics [77, 65].

2.3.4 Open Challenges and Research Gaps

Despite significant progress, several challenges remain in CP solver configuration:

1. **Solver-Agnostic Configuration**: Most existing approaches are tailored to specific solvers. There is a need for general, solver-agnostic configuration frameworks that can handle the diverse parameter spaces of different CP solvers while providing a unified interface [16].

2. **Handling Complex Parameter Dependencies**: Current methods struggle with complex conditional parameter dependencies that are common in CP solvers. Better methods are needed for modeling and exploring these hierarchical spaces [52].

3. **Transfer Learning and Meta-Learning**: Leveraging knowledge from previously solved configuration problems to accelerate new configurations remains underexplored in CP. Meta-learning approaches that learn across problem classes could significantly reduce configuration overhead [102].

4. **Interpretable Configuration**: While current methods can find high-performing configurations, they often provide little insight into why certain configurations work well. More interpretable configuration methods could help users understand solver behavior and make informed decisions [51].

5. **Integration with Declarative Modeling**: There is a disconnect between configuration methods and declarative modeling environments like MiniZinc. Better integration could enable automatic configuration based on model characteristics rather than just instance features [76].

2.3.5 Performance Evaluation in CP Configuration

Evaluating configuration methods for CP solvers requires careful consideration of several factors:

- **Benchmark Selection**: Representative benchmarks should cover diverse problem classes and difficulty levels. Common benchmarks include those from the MiniZinc Challenge [99], CP challenges, and application-specific problem sets.

- **Evaluation Metrics**: Beyond simple runtime, metrics should consider solution quality (for optimization problems), anytime performance, robustness across instances, and configuration cost (time spent tuning).

- **Statistical Significance**: Given the stochastic nature of both configuration methods and solver runs, statistical tests (e.g., Wilcoxon signed-rank tests, critical difference diagrams) are essential for drawing reliable conclusions [30].
- **Baseline Comparisons**: Comparisons should include appropriate baselines such as default configurations, random search, and established configuration tools like SMAC or irace.

2.4 Multi-Objective Optimization

2.4.1 Formal Definition and Notations

We consider $k \geq 2$ objectives to be minimized,

$$f(x) \;=\; \Big(f_1(x),\, \ldots,\, f_k(x)\Big),$$

over a feasible set $\mathcal{S}(\mathcal{C})$ defined by constraints $\mathcal{C}$.

Pareto dominance. For two feasible solutions $x, y \in \mathcal{S}(\mathcal{C})$, we say that x *Pareto-dominates* y (written $x \prec y$) if

$$\forall i \in \{1, \ldots, k\} :\; f_i(x) \leq f_i(y) \quad \text{and} \quad \exists j \in \{1, \ldots, k\} :\; f_j(x) < f_j(y).$$

The set of nondominated solutions is the *Pareto set* $\mathcal{P}$, and its image $f(\mathcal{P})$ is the *Pareto front*.

CP-based MOO approaches. In CP, we either *scalarize* (e.g., weighted sums or ε-constraints) or *search directly* for diverse nondominated solutions within a budget [27, 44].

2.4.2 Multi-Objective Optimization in Algorithm Configuration

While most algorithm configuration studies optimize a single measure (e.g., runtime), real-world optimization often involves multiple conflicting goals. In CP, balancing runtime, solution quality, and robustness requires multi-objective approaches. Evolutionary algorithms such as NSGA-II [26] and indicator-based methods like hypervolume maximisation [107] provide the foundations for evaluating Pareto-optimal solver configurations.

Integrating these indicators with BO has recently produced efficient multi-objective HPO algorithms (MOBO) [1, 39]. These tools can help users understand the trade-offs in a solver's

performance (e.g., speed vs. solution quality). In CP, this multi-objective perspective remains underdeveloped, but the potential to measure solver behaviour holistically aligns well with decision-support applications in AEC and engineering design.

Common solution concepts include: (i) ε-dominance and reference-point methods for controlled trade-offs, (ii) scalarization families to probe different regions of the front, and (iii) diversity-aware selection to maintain spread. When solver calls are expensive, surrogate-assisted strategies and restart schedules improve coverage within strict time limits [48].

2.4.3 Performance Indicators and Solution Scenarios

We report dominated hypervolume, inverted generational distance, coverage (C-metric), and the cardinality/diversity of the obtained set. In budgeted CP settings, anytime hypervolume traces, best-known trade-offs at checkpoints, and reproducibility across seeds/instance subsets are also informative [6].

Scenarios range from *design exploration* (quickly obtain a diverse Pareto set), to *decision support* (refine trade-offs near a region of interest), to *certification* (meet multiple thresholds simultaneously). The choice of modeling (e.g., global constraints, surrogate assistance) and search strategy (e.g., restarts, portfolios) should reflect the scenario and evaluation budget [44, 48].

Part II

Solver Intelligence: Hyperparameter Optimization for Constraint Programming Solvers

Chapter 3

The Probe and Solve Algorithm Framework

Contents

3.1 Introduction

CP provides a declarative framework for modeling and solving combinatorial problems, where solver performance critically depends on the *search strategy*, specifically, the heuristics for variable and value selection. Despite numerous proposed heuristics, no single strategy demonstrates universal effectiveness across problem types [95, 100]. Manual strategy selection

requires extensive expertise and trial-and-error, creating a significant barrier to effective CP usage.

To address this challenge, we introduce *PSA*, a comprehensive framework that automates solver configuration through hyperparameter optimization, with a primary focus on search strategy selection. The PSA framework presented in this chapter represents the culmination of research documented in multiple publications [45, 46]. While early versions established the core two-phase architecture, subsequent work introduced adaptive timeout mechanisms, stagnation detection, memory caching, and a modular component design. This chapter presents the complete, integrated framework that synthesizes these developments into a unified system.

3.1.1 Motivation and Core Idea

The core insight behind PSA is that search strategy selection can be treated as a hyperparameter optimization problem. Instead of relying on fixed default strategies or manual tuning, PSA systematically explores different strategy combinations during a dedicated probing phase, then uses the best-found strategy for the actual solving. This approach balances exploration (finding good strategies) with exploitation (using them to solve the problem).

Research Questions. This chapter addresses:

- **RQ1:** Can HPO methods be effectively applied to the discrete and complex hyperparameter spaces of CP solvers, and how can they be used to operate efficiently in this context?
- **RQ2:** Among different HPO paradigms, model-free (e.g., random search, Hyperband) and model-based (e.g., BO), which approaches are best suited for configuring CP solvers under realistic time constraints?
- **RQ3:** How can the approach be made fully parameterless while maintaining robustness across different allocated times?

Contributions. The main contributions are:

1. Formalization of search-strategy selection as HPO over (S_{var}, S_{val})
2. Complete PSA framework with two-phase architecture and adaptive mechanisms
3. Modular design with configurable components for HPO methods, time allocation, timeout evolution, and stopping conditions

4. Memory-based caching to avoid redundant configuration evaluations
5. Detailed algorithmic specifications for probing, solving, and integrated operation

Chapter Structure. Section 3.2 formalizes the problem. Section 3.3 overviews PSA. Sections 3.4 and 3.5 detail the phases. Section 3.6 presents integration. Section 3.7 describes components. Section 3.8 discusses publications.

3.2 Problem Formalization

Let S_{var} denote variable selection strategies (e.g., `first_fail`, `dom/wdeg`, `impact`) and S_{val} denote value selection strategies (e.g., `indomain_min`, `indomain_max`, `indomain_random`). A complete search strategy is $\lambda = (v, w) \in \Lambda = S_{var} \times S_{val}$.

Given a COP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, f)$ and global timeout T_g, find $\lambda^* \in \Lambda$ optimizing performance:

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \mathbb{E}_{I \sim \mathcal{D}}[m(A(\lambda, I, T_g))]$$

where $m(\cdot)$ measures performance (e.g., runtime, solution quality). This treats strategy selection as HPO with Λ as hyperparameter space.

3.2.1 Interpretation of the Formalization

This formulation captures several important aspects of the problem:

- The strategy space Λ is discrete and combinatorial in nature
- Each evaluation of $m(A(\lambda, I, T_g))$ involves running the solver, which can be computationally expensive
- The expectation over instances $\mathbb{E}_{I \sim \mathcal{D}}$ indicates we seek robust strategies that work well on average
- The time budget T_g imposes practical constraints on how much exploration is feasible

3.3 PSA Framework Overview

To systematically optimize a set of hyperparameters for CP solvers, we introduce PSA, a flexible and adaptive framework designed to make effective use of a limited time budget. The core idea of PSA is to partition a global time-limit (T_g) into two distinct phases:

1. A *probing phase*, where a dedicated portion of the time budget is used to explore a wide range of hyperparameters. This is achieved by running many short-lived, time-limited solver runs, each with a different configuration, to gather performance data.

2. A *final solving phase*, where the single best-performing configuration identified during probing is used to solve the problem instance with the entire remaining time budget.

This two-phase approach provides a structured balance between the exploration of the hyperparameter space and the exploitation of the most promising strategy found. The framework is designed to be highly modular, allowing different strategies for time management, hyperparameter selection, and timeout evolution to be composed, enabling a thorough investigation of their combined effects.

The PSA framework follows a two-phase architecture that partitions the global time budget T_g into probing time t_p and solving time t_s:

$$t_p = \rho \cdot T_g, \quad t_s = T_g - t_p, \quad \rho \in [0, 1]$$

With $\rho = 0.2$ as the default value based on empirical analysis, 20% of the total time is dedicated to exploring strategies, while 80% is reserved for actual solving with the best strategy found. This balanced allocation addresses the fundamental trade-off between exploration and exploitation in algorithm configuration [46].

Figure 3.1 illustrates the PSA architecture. The framework begins with a COP instance and global time budget, proceeds through the probing phase to identify the best configuration λ^*, and concludes with the solving phase using the remaining time.

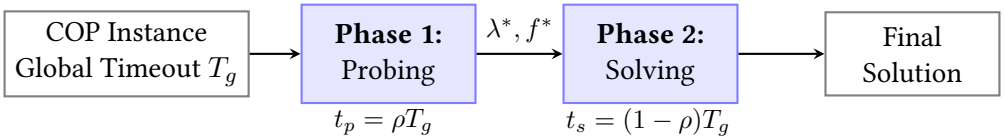


Figure 3.1 PSA two-phase architecture with time allocation. The framework partitions the global time budget T_g into probing time t_p (for exploring hyperparameters) and solving time t_s (for execution with the best configuration).

3.4 The Probing Phase

The goal of the probing phase is to efficiently sample the hyperparameter space to identify a high-quality configuration. The phase consists of a sequence of iterative trials, hereafter referred to as *rounds*. In each round, a single set of hyperparameters is selected and evaluated

by running the solver for a short duration. The execution of the probing phase is detailed in Algorithm 1 and is controlled by these key orthogonal strategy types.

3.4.1 Key Features of the Probing Phase

The probing phase incorporates several sophisticated mechanisms to optimize exploration efficiency:

3.4.1.1 Memory-Based Caching

A memory array $\mathcal{M}$ stores previously evaluated configurations and their results. When a configuration is revisited, the solver retrieves the stored results instead of performing redundant evaluations. This accelerates the optimization process and avoids redundant computation, particularly for large-scale problem instances.

3.4.1.2 Global Time Management

This high-level component determines the total time budget allocated to the probing phase. We implement two approaches:

- **Static percentage allocation:** A fixed percentage of T_g is reserved for the entire probing phase. For example, with a T_g of 1800 seconds and a 20% allocation, the probing phase will run for a total of 360 seconds before automatically transitioning to the solving phase. This ensures a predictable and consistent time split across different experiments.

- **Iteration-limited allocation:** The probing phase continues until either the T_g is exhausted or a predefined maximum number of configuration trials (`max_tries`) has been completed. This strategy allows for more flexible exploration, as the total probing time is determined by the cumulative runtime of the trials, making it suitable for scenarios where individual probe runtimes are highly variable.

3.4.1.3 Adaptive Timeout Initialization

This component determines the timeout for each individual configuration evaluation within the probing phase. This sets the time limit for the first configuration to be tested.

- **Static initial timeout:** A fixed, small timeout (e.g., 5 seconds) is used. This provides a consistent baseline but may be too short or too long for certain problem instances.

Algorithm 1 Probing Phase of PSA

Input: COP $(\mathcal{X}, \mathcal{D}, \mathcal{C}, f)$; HPO $\mathcal{H}$; Λ; T_g; τ_0; $TimeInit$; $Evolve$; $stop\_type$; L.
Output: $\lambda^\star$, $f^\star$, t_{rem}.

$t_p, t_s \leftarrow \text{AllocateTime}(T_g)$ *Probe/solve budgets*
Initialize: $f^\star \leftarrow \infty$, $\lambda^\star \leftarrow$ default, $stag \leftarrow 0$, $t_0 \leftarrow \text{Now}()$

if $TimeInit = \textsc{FirstRuntime}$ **then** *Adaptive timeout*
    $(sol, rt) \leftarrow \text{Solve}(\text{COP}, \text{default}, t_p)$
    $\tau \leftarrow rt$ *Use runtime*
    if sol **then** $f^\star \leftarrow f(sol)$; $\lambda^\star \leftarrow$ default
else *Static timeout*
    $\tau \leftarrow \tau_0$

while $\text{Now}() - t_0 < t_p$ **do** *Main loop*
    $\lambda \leftarrow \mathcal{H}.\text{NextConfig}()$
    if $\lambda = \varnothing$ **then break**

    if $\lambda \in \mathcal{M}$ **then** *Check memory array*
        $(obj, rt) \leftarrow \mathcal{M}[\lambda]$ *Retrieve cached result*
    else
        $(sol, rt) \leftarrow \text{Solve}(\text{COP}, \lambda, \tau)$
        $obj \leftarrow f(sol)$ if sol else ∞
        $\mathcal{M}[\lambda] \leftarrow (obj, rt)$ *Store in memory array*

    if $obj < f^\star$ **then** *Better solution found*
        $f^\star \leftarrow obj$; $\lambda^\star \leftarrow \lambda$; $stag \leftarrow 0$
    else
        $stag \leftarrow stag + 1$

    $\mathcal{H}.\text{UpdateModel}(\lambda, obj, rt)$
    $\tau \leftarrow Evolve(\tau)$ *Update timeout*

    if $stop\_type = \textsc{FirstSolution}$ and $obj \neq \infty$ **then break**
    if $stop\_type = \textsc{Stagnation}$ and $stag \geq L$ **then break**

$t_{\text{rem}} \leftarrow T_g - (\text{Now}() - t_0)$
return $(\lambda^\star, f^\star, t_{\text{rem}})$

- **First-runtime timeout:** The framework first performs a preliminary run of the solver using its default configuration. This run is allocated an informed time limit which is up to the total available probe budget, and will stop as soon as the first solution is found. The actual runtime observed from this initial run is then used as the timeout for subsequent probes, thereby adapting the timeout to the intrinsic difficulty of the problem instance.

3.4.1.4 Timeout Evolution Patterns

After the initial round, the timeout for subsequent rounds can evolve to adapt to the performance of previously tested configurations. Our framework manages this through a modular *Timeout Evolution Strategy*, which implements an `Evolve(timeout)` function to compute the timeout for the next round based on the value from the current one.

We implement and test three concrete instances of this strategy:

- **Static evolution:** This strategy's `Evolve` function simply returns the same `timeout` it was given, keeping the timeout fixed.
- **Geometric evolution:** Here, the $\texttt{Evolve}(t_i)$ function returns the current timeout multiplied by a predefined growth factor, $\beta > 1$. For example, $t_{i+1} = \texttt{Evolve}(t_i) = t_i \times \beta$.
- **Luby sequence evolution:** This strategy's $\texttt{Evolve}(t_i)$ function maintains an internal counter k. It returns $t_{i+1} = t_0 \cdot \text{Luby}(k)$, where t_0 is the initial timeout and $\text{Luby}(k)$ is the k-th value in the sequence (1, 1, 2, 1, 1, 2, 4, ...), incrementing k after each call.

3.4.1.5 Intelligent Stopping Conditions

The probing phase is allocated a specific time budget (e.g., a percentage of the global time limit or up to a maximum number of tries). The phase will terminate when this budget is exhausted, but it can also be configured to stop early to pivot to the final solving phase more quickly. The defined stopping conditions are:

- **Timeout**: The primary stopping condition is the exhaustion of the allocated probing time budget. The phase will always terminate when its time is up, ensuring that the final solving phase receives its planned portion of the global time limit.
- **First solution found:** The probing phase can be configured to halt as soon as any configuration finds the first feasible solution. This is particularly useful in scenarios where quickly finding any valid solution is more critical than extensive exploration for the absolute best configuration.

- **Stagnation:** The process can terminate if the best-found solution has not improved after a predefined number of rounds. This prevents the framework from wasting time on continued exploration if the HPO strategy appears to have converged or is no longer making progress.

3.4.1.6 HPO Method Interface

This is the core HPO engine of the PSA, responsible for selecting the next set of hyperparameters to evaluate. To ensure modularity, the framework requires any HPO strategy to implement a specific interface with three key functions:

1. **`NextConfig()`:** Returns the next set of hyperparameters to be evaluated from the search space.

2. **`UpdateModel(params, runtime)`:** Takes the parameters of the last trial and its resulting runtime and objective to update the strategy's internal model. For model-free methods like Hamming distance, this may simply involve updating the incumbent.

3. **`AllocateTime(`T_g`)`:** Partitions the global timeout into probing and solving budgets:

$$\texttt{AllocateTime}(T_g) = (\rho T_g,\ (1-\rho)T_g)$$

   where ρ is the probing ratio (default $\rho = 0.2$). This creates a clear separation: t_p for configuration exploration, t_s for solving with the best configuration.

4. **`GetBestConfig()`:** Returns the best configuration found so far, which is used in the final solving phase.

We implement BO as a core component of PSA. It is a concrete strategy that conforms to this interface. BO is a sophisticated model-based strategy where `NextConfig` uses an acquisition function over a GP model to choose the next point. `UpdateModel` retrains the GP with the new data point.

3.5 The Solving Phase

Once the probing phase concludes, PSA transitions to the solving phase, which utilizes the best-found configuration to solve the instance with the remaining time budget.

The solving phase operates as follows:

Algorithm 2 Solving Phase of PSA

Input: COP $(\mathcal{X}, \mathcal{D}, \mathcal{C}, f)$; HPO $\mathcal{H}$; $\lambda^\star$; $f^\star$; t_{rem}.
Output: sol_{final}, $f(sol_{\text{final}})$.

if $f^\star = \infty$ **then** *No solution found in probing*
    $\lambda_{\text{final}} \leftarrow \mathcal{H}$.GetBestConfig()
    $(sol_{\text{final}}, rt) \leftarrow \text{Solve}(\text{COP}, \lambda_{\text{final}}, t_{\text{rem}})$
else *Incumbent exists; try to improve*
    $\mathcal{C}' \leftarrow \mathcal{C} \cup \{f(\mathcal{X}) < f^\star\}$ *Add cut constraint*
    $(sol_{\text{final}}, rt) \leftarrow \text{Solve}((\mathcal{X}, \mathcal{D}, \mathcal{C}', f), \lambda^\star, t_{\text{rem}})$

return $(sol_{\text{final}}, f(sol_{\text{final}}))$

1. The single best-performing set of hyperparameters ($\lambda^\star$) identified during the probing phase is retrieved.

2. If any solution was found during probing (i.e., $f^\star < \infty$), a new objective cut constraint is added to the model: $\mathcal{C}' \leftarrow \mathcal{C} \cup \{f(\mathcal{X}) < f^\star\}$. This constraint forces the solver to search only for solutions that are strictly better than the current best, implementing an optimization loop that focuses the search on improvement rather than re-finding known solutions.

3. The solver is executed with $\lambda^\star$ and the (potentially updated) constraint model $\mathcal{C}'$, using the entire remaining time budget (T_g − elapsed time from probing).

3.5.1 Fallback Mechanism

If no solution was found during probing, the solving phase uses the best configuration predicted by the HPO method's internal model, providing robustness even when initial exploration fails to find feasible solutions.

3.6 Integrated PSA Framework

Algorithm 3 integrates both phases into a complete framework that operationalizes the two-phase tuning strategy.

3.6.1 Framework Properties

The integrated PSA framework exhibits several desirable properties:

Algorithm 3 Complete PSA Framework

Input: COP $(\mathcal{X}, \mathcal{D}, \mathcal{C}, f)$; T_g; ρ; HPO $\mathcal{H}$; Λ.
Output: sol_{final}, $f(sol_{\text{final}})$.

$t_p \leftarrow \rho \times T_g$ ..... *Probing budget*
$t_s \leftarrow T_g - t_p$ ..... *Solving budget*

Phase 1: Probing
$(\lambda^\star, f^\star, t_{\text{rem}}) \leftarrow \text{ProbingPhase}(\text{COP}, \mathcal{H}, \Lambda, T_g, \tau_0, \text{TimeInit}, \text{Evolve}, \text{stop\_type}, L)$. *Alg. 1*

Phase 2: Solving
$(sol_{\text{final}}, f_{\text{final}}) \leftarrow \text{SolvingPhase}(\text{COP}, \mathcal{H}, \lambda^\star, f^\star, t_s)$ ..... *Alg. 2*

return $(sol_{\text{final}}, f_{\text{final}})$

- **Time-awareness:** Strictly respects the global timeout T_g, ensuring practical applicability.
- **Adaptivity:** Adjusts behavior based on problem difficulty through timeout mechanisms.
- **Robustness:** Includes fallback mechanisms ensuring solution return even if individual phases fail.
- **Modularity:** Components (HPO methods, time allocation strategies) can be easily swapped.
- **Resource efficiency:** Memory caching minimizes redundant computations.

3.7 Modular Components

PSA's modular design enables systematic evaluation of different design choices and adaptation to specific problem characteristics. Table 3.1 summarizes the main components.

3.7.1 Component Interactions

The modular components work together through well-defined interfaces:

- The **HPO method** guides configuration selection based on its internal model.
- The **time allocation** strategy determines how much time is available for exploration versus exploitation.

Component	Description and Options
HPO Method	Strategy for selecting configurations: *Bayesian Optimization* (model-based), *Hamming Distance Search* (local search)
Time Allocation	Division of global timeout: *Static percentage* (fixed ρ), *Dynamic allocation* (adaptive based on progress)
Timeout Evolution	Pattern for adjusting per-strategy timeout: *Static* (constant), *Geometric* ($t_{i+1} = \beta t_i$), *Luby sequence* (theoretical restart sequence)
Stop Condition	Criteria for early termination: *Timeout* (budget exhausted), *First Solution* (any feasible solution), *Stagnation* (no improvement for L rounds)
Memory Array	Cache storing evaluated configurations and results to avoid redundant computations
Solver Interface	Abstraction layer enabling integration with different constraint solvers (ACE, Choco, etc.)

Table 3.1 PSA modular components

- The **timeout evolution** pattern controls the depth of individual configuration evaluations.
- The **stop conditions** provide early termination mechanisms to reallocate resources.
- The **memory array** accelerates the process by caching previous evaluations.

This modular architecture enables:

- Easy comparison of different HPO methods
- Adaptation to specific problem characteristics
- Extension with new components without modifying core logic
- Systematic evaluation through controlled experiments

3.8 Published Work

The PSA framework represents the unified theoretical framework that underlies our previously reported experimental studies:

- **HADDAD, H., TALBOT, P., & BOUVRY, P.** (2024). *Comparison of Hyperparameter Optimization Methods for Selecting Search Strategy of Constraint Programming Solvers.* PTHG-24: The Seventh Workshop on Progress Towards the Holy Grail, Girona, Spain.

- **HADDAD, H., TALBOT, P., & BOUVRY, P.** (2024). *Selecting Search Strategy in Constraint Solvers using Bayesian Optimization.* 2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI), Herndon, VA, USA.

These publications document the framework's evolution from initial concept to the complete modular system presented in this chapter. The integrated framework synthesizes insights from all three studies into a unified approach for automated solver configuration.

3.9 Summary

This chapter introduces the complete PSA framework for automated hyperparameter optimization in constraint programming solvers. While preliminary applications and experimental results have been reported in prior publication [45, 46], this chapter presents the complete theoretical formulation, unified algorithmic specification, and comprehensive design rationale that constitute the core scientific contribution of this thesis.

Key innovations include:

1. Two-phase architecture balancing exploration and exploitation
2. Memory-based caching to avoid redundant evaluations
3. Adaptive timeout mechanisms adjusting to problem difficulty
4. Modular component design enabling systematic evaluation
5. Intelligent stopping conditions for efficient resource allocation

The PSA framework provides a practical, automated approach for improving constraint solver performance without requiring expert knowledge or manual parameter tuning. In the next chapter, we empirically evaluate PSA across multiple benchmarks, solvers, and configuration spaces.

Chapter 4

Experimental Evaluation and Analysis of PSA

Contents

4.1 Introduction

This chapter presents a comprehensive empirical validation of the PSA framework detailed in Chapter 3. The primary objective is to evaluate PSA's performance, robustness, and generalization across a wide range of problem types, solvers, and hyperparameter optimization configurations.

The experimental work is structured around two complementary evaluations designed to systematically address our research questions:

- **Section 4.2: PSA on Restricted Hyperparameter Spaces** investigates PSA's core effectiveness using a focused set of hyperparameters—variable and value selection heuristics—that are common across different solvers. This establishes a controlled baseline for comparing HPO methods and understanding fundamental solver behavior.
- **Section 4.3: PSA on Complete Solver Configurations** evaluates the complete, modular PSA architecture against the full hyperparameter spaces of modern solvers. This large-scale study systematically analyzes PSA's internal components (time allocation, timeout evolution, stopping conditions) to identify optimal configurations and understand their interactions.

Research Questions. The experiments detailed in this chapter are designed to answer the following research questions:

- **RQ4:** To what extent does this approach generalize across solvers and modeling environments, and how consistent is its performance under varying computational budgets?
- **RQ5:** How can solver performance be systematically benchmarked across different HPO strategies to identify their strengths and limitations?

Contributions and Scope. The main empirical contributions of this chapter are as follows:

1. We provide a comprehensive empirical validation of PSA on a diverse suite of benchmark problems, demonstrating its effectiveness and robustness.

2. We demonstrate that a fixed default probing ratio (e.g., 20%) provides a strong balance between exploration and efficiency, enabling PSA to operate in a nearly parameterless fashion.

3. We conduct an analytical study of solver behavior, revealing how high-level hyperparameters influence solver search dynamics and how the effectiveness of different PSA strategies depends strongly on the type of problem.

4. We present the first integration of the ACE solver into the CPMpy library, a technical contribution enabling the advanced benchmarking experiments conducted in this work.

Chapter Structure. The remainder of this chapter is organized as follows. Section 4.2 presents the evaluation of PSA with restricted hyperparameter spaces across multiple solvers. Section 4.3 details the large-scale benchmarking of PSA with complete solver configurations. Finally, Section 4.4 provides a unified discussion of all experimental findings and concludes the chapter.

4.1.1 Experimental Setup

The experiments presented in this chapter were carried out using the high-performance computing (HPC) facilities of the University of Luxembourg [103] (see `https://hpc.uni.lu`). The Aion cluster compute nodes used in our experiments are equipped with 2× AMD EPYC Rome 7H12 processors at 2.6 GHz (64 cores each) and 256 GB RAM.

4.2 PSA on Restricted Hyperparameter Spaces

4.2.1 Motivation and Experimental Goals

This evaluation focuses on PSA's core capability: automatically selecting search strategies from the variable and value heuristics available in CP solvers. We restrict the hyperparameter space to these fundamental components that are common across solvers, allowing us to:

- Compare different HPO methods (random search, Hyperband, Bayesian optimization) for solver configuration under realistic time constraints.

- Determine the optimal time allocation between probing and solving phases.

- Validate PSA's robustness across different problem types and time budgets.

- Assess generalization across solvers (ACE, Choco) and modeling frameworks (XCSP3, MiniZinc).
- Compare dynamic versus static search strategies within a controlled experimental setting.

4.2.2 Experimental Configuration

Our study utilizes data from the fifth international XCSP3 constraint solver competition conducted in 2023 [67]. We specifically target all the COP instances derived from this competition, totaling 250 instances that cover a variety of constraints and objective functions. For each problem, we align our global timeout setting of 1200 seconds with the one utilized by the XCSP3 challenge.

To validate that PSA produces encouraging results, we have opted to set the probing phase timeout to different percentages of the global timeout: 5%, 10%, 20%, 50%, and 100%. This enables us to compare the efficiency and applicability of different timeout budgets.

4.2.3 Restricted Hyperparameter Configuration

We focus on hyperparameters that have direct analogs in most CP solvers: variable selection heuristics (S_{var}) and value selection heuristics (S_{val}). For the ACE solver, we evaluate three configurations:

- **Full heuristic space:** All heuristics:
  Variables: `RunRobin,Wdeg,Memory,PickOnDom,FrOnDom,WdegOnDom,ProcOnDom,Regret,FrbaOnDom,Ddeg`
  Values: `Dist,OccsR,Median,AsgsFp,Flrs,Bivs,First,AsgsFm,Last,Robin,RunRobin,Bivs2,InternDist,Occs,FlrsE`
- **Simple static subset:**
  Variables: `Deg,Rand,Lexico,Srand`
  Values: `Vals,First,Last,RunRobin,Robin,Srand,Rand`
- **Dynamic heuristic subset:**
  Variables: `PICK3, CACD, FRBA` with all value heuristics.

4.2.4 Search Strategies

Our framework focuses specifically on search strategy configuration for several important reasons:

1. **Cross-Solver Comparability:** Different constraint solvers (e.g., Gecode, Choco, OR-Tools) have diverse architectures and internal parameters. Search strategies represent a *common denominator*—a hyperparameter space that exists across most CP solvers, enabling fair comparison and transferable insights.

2. **High Impact:** Search strategies have been empirically shown to have substantial influence on solver performance, often more than low-level solver parameters [78]. The choice of variable and value selection heuristics can lead to orders-of-magnitude differences in runtime.

3. **User Accessibility:** Unlike internal solver parameters that require deep implementation knowledge, search strategies are part of the CP modeling interface. Users routinely specify search strategies, making this a practical target for automation.

4. **Discrete Combinatorial Space:** The space of search strategies forms a well-defined discrete combinatorial structure ($S_{var} \times S_{val}$), making it amenable to systematic exploration via HPO methods.

While our framework could be extended to other solver parameters, we concentrate on search strategies as they provide the most immediate practical value while maintaining methodological rigor for comparative studies.

4.2.5 Implementation Details

Our approach is implemented in Python, leveraging several libraries for optimization and CP. The key libraries used include scikit-optimize (`skopt`) [93] for HPO, MiniZinc and PyCSP3 libraries for CP.

The search strategies of the solvers are described declaratively in a JSON file. This allows us to easily modify and experiment with different sets of available strategies without changing the core code.

For instance, a simplified example of the ACE solver within the XCSP3 looks like this:

```
"XCSP3": {
 "Search-Strategy": {
  "Varh_values":
   ["PickOnDom", "FrbaOnDom", "WdegOnDom"],
  "Valh_values":
   ["First", "Median"]}}
```

In this example, `PickOnDom`, `FrbaOnDom` and `WdegOnDom` are the variable selection strategies, while `First` and `Median` are the value selection strategy.

4.2.6 Results and Analysis

4.2.6.1 Random Search Performance

In this section, we evaluate the performance of PSA when random search is employed as the HPO approach. Our focus is on comparing the results of PSA with baseline performances, considering all variable selection strategies and value selection strategies provided by the ACE solver as hyperparameters.

Upon applying random search as the HPO approach, it is observed that the results are not as robust as the ones obtained using the baselines. When considering the random search with a ratio of 0.2, it is observed that the results vary across different strategies. For instance, PSA outperformed the CACD strategy in 18.07% of the instances, while CACD achieved a superior objective of 23.11% of the cases. Similarly, PSA surpassed the FRBA strategy in 15.04% of the cases, while FRBA attained a higher objective of 22.36% of the instances, and for the rest of them, both approaches yielded equivalent results.

This can be attributed to several factors, the most significant being the inherent randomness of the approach. Despite this, random search remains a popular choice in the HPO field due to its simplicity and low computational overhead, which ensures minimal system strain during execution.

The results of PSA with random search at a probing ratio 0.2 are illustrated in Figure 4.1a.

4.2.6.2 Hyperband Search Performance

For the second step, we assessed the performance of PSA when Hyperband search is utilized as the HPO method taking into account all variable selection strategies and value selection strategies mentioned above. Hyperband search is a resource-efficient variant of random search that uses a bandit-based approach to allocate resources to different configurations.

When employing Hyperband search with a probing ratio of 0.2, the results demonstrate variability across different strategies. For instance, PSA outperformed the PICK3 strategy in 17.01% of the instances, while PICK3 achieved a superior objective in 22.45% of the instances. This indicates that while the Hyperband method can effectively handle a large number of hyperparameters, the performance can vary significantly depending on the specific strategy employed and the complexity of the problem at hand.

The results of PSA with Hyperband search are illustrated in Figure 4.1b. This figure provides insights into the performance of PSA with Hyperband search compared to the baseline strategies, thereby offering a comprehensive view of the effectiveness of different HPO methods in enhancing the solver's performance.

4.2.6.3 Bayesian Optimization Performance

In the next step, we examined the performance of PSA when BO is employed as the HPO method. Upon implementing BO as the HPO approach, we anticipate a different set of results compared to random search and Hyperband search.

It is observed that PSA outperforms the default solver in 29.49% of the models. Interestingly, in 44.87% of the models, PSA and the baselines produce identical results, suggesting that no single strategy consistently excels across all situations.

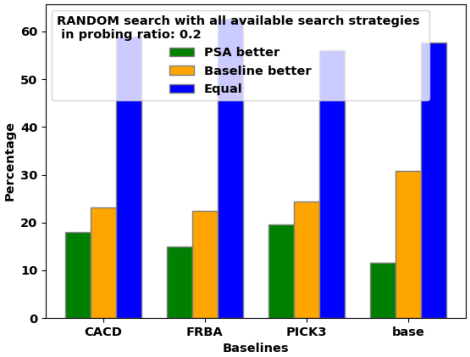
When employing BO with a probing ratio of 0.2, the results demonstrate variability across different strategies. For instance, PSA outperformed the FRBA strategy in 25.20% of the instances, while FRBA achieved a superior objective of 19.92% of the instances. Indeed, the results suggest that employing BO with a probing ratio of 0.2 can yield better results than all the baselines. This demonstrates the potential of BO as an effective HPO approach, particularly when dealing with complex problems with a large number of hyperparameters.

The results of PSA with BO at a probing ratio 0.2, are illustrated in Figure 4.1c.

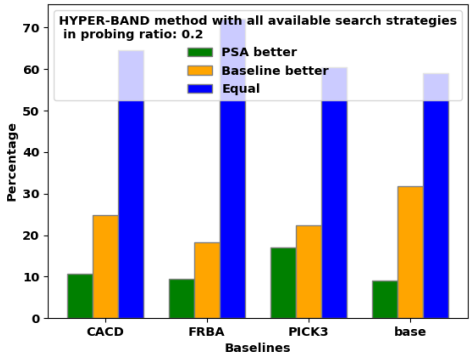
4.2.7 Detailed Analysis and Discussion

The detailed results obtained from the various HPO methods and different probing ratios are presented in Table 4.1. We conducted tests using a range of probing ratios, specifically 5%, 10%, 20%, 50%, and 100% of the global timeout. Our results indicated that, for our specific set of problems, a probing ratio of 20% generally performed better than the others. Also, this table incorporates two additional columns: *Fallback to Default* and *Same Search Strategy*. The *Fallback to Default* column measures the instances where PSA could not determine a search strategy within the given timeframe, leading to the use of the default search strategy during the solving phase. The *Same Search Strategy* column signifies the instances where PSA identified a search strategy identical to the solver's default search strategy post the probing phase. These additional insights enhance our understanding of PSA's performance and decision-making process.

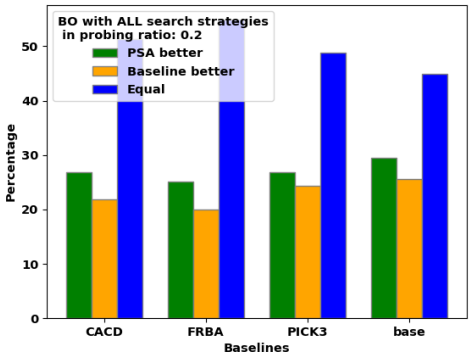
Furthermore, it is important to note that unlike other approaches which are invasive and require modifications to the solver's code, our approach is non-invasive and generic, capable



(a) Random search with all available S_{var} and S_{val}



(b) Hyperband with all available S_{var} and S_{val}



(c) BO with all S_{var} and S_{val}

Figure 4.1 Comparative performance of PSA using different HPO methods at a probing ratio of 0.2.

of working with any solver. Also, it can dynamically change some crucial parameters of HPO, eliminating the need for their initialization beforehand. This addresses a significant limitation as it allows PSA to adaptively set the number of rounds and timeout for each round, offering a more flexible and efficient approach. For instance, in contrast to the static nature of classical BO, our approach can dynamically adjust these parameters based on the problem's complexity and the solver's performance, thereby enhancing the overall efficiency and effectiveness of the solver.

In our study, we explored various solver parameters and strategies. Here, we discuss our findings:

Search Strategies with Restricted Subset of Variables and Value Selection Strategies. We implemented different subsets of search strategies as hyperparameters for PSA to see if PSA with static search strategies could outperform the dynamic strategies. Our results indicate that dynamic approaches surpass static approaches in this context.

Solver Parameters. We also examined different restart annotations such as Luby sequence restart and geometric restart options, including the number of nodes explored for restarting [47]. We also explored different solver parameters, such as the number of last conflict options. However, due to the vast number of options and limited global timeout, not all of these options were beneficial for our specific approach. In some cases, they even worsened the approach. This could be attributed to the state-space becomes too large and the probing phase becomes too short to find a good configuration.

4.2.8 Robustness and Generalization Analysis

4.2.8.1 Validating Robustness in Shorter Timeouts

In our study, we employed two statistical methods, Spearman's rank correlation [31] and Kendall's tau [86], to analyze the results of the probing phase. The aim was to determine whether the algorithm could identify effective search strategies based on the rankings observed at both short and global timeouts, and to understand how these rankings at shorter timeouts correlate with those obtained at the global timeout.

We randomly selected three instances and ran them 10 times, each time recording the intermediate results and ranking the search strategies. For each run, we separated the ranking of the results after different timeouts: 5%, 10%, 20%, 50%, and 100% of the global timeout. We then examined whether the rankings of the results, evaluated based on the objective function, were correlated. This allowed us to investigate whether the behavior of the search strategies remained consistent across different runs and for the specific problem.

Method		Baselines	Results (%)			Additional Results (%)	
			PSA Better	Baselines Better	Equal Results	Fallback to default	Same search strategy
Random Search All available S_{var} and S_{val}	0.05	CACD	19.67	**22.59**	57.74	24.70	0.00
		FRBA	19.03	**21.46**	59.51		
		PICK3	**26.32**	23.08	50.61		
		Default	12.66	**26.58**	60.76		
	0.1	CACD	16.81	**25.21**	57.98	17.89	0.41
		FRBA	16.26	**23.98**	59.76		
		PICK3	19.51	**26.42**	54.07		
		Default	15.38	**21.79**	62.82		
	0.2	CACD	18.07	**23.11**	58.82	13.01	0.81
		FRBA	15.04	**22.36**	57.32		
		PICK3	19.51	**24.39**	56.10		
		Default	11.54	**30.77**	57.69		
	0.5	CACD	**21.70**	16.98	61.32	11.82	0.00
		FRBA	15.00	**18.18**	66.82		
		PICK3	**20.91**	19.55	59.55		
		Default	17.46	**19.05**	63.49		
	1.0	CACD	1.67	**13.33**	85.00	15.27	0.00
		FRBA	1.56	**13.28**	85.16		
		PICK3	0.76	**14.50**	84.73		
		Default	2.33	**6.98**	90.70		
Hyperband method All available S_{var} and S_{val}	0.05	CACD	12.66	**28.69**	58.65	25.30	0.81
		FRBA	11.84	**30.20**	57.96		
		PICK3	19.18	**31.84**	48.98		
		Default	10.26	**26.92**	62.82		
	0.1	CACD	13.87	**31.09**	55.04	17.48	0.81
		FRBA	11.79	**29.27**	58.94		
		PICK3	15.04	**32.11**	52.85		
		Default	12.82	**33.33**	53.85		
	0.2	CACD	10.64	**24.82**	64.54	21.77	0.68
		FRBA	9.52	**18.37**	72.11		
		PICK3	17.01	**22.45**	60.54		
		Default	9.09	**31.82**	59.09		
	0.5	CACD	9.38	**22.66**	67.97	17.91	0.75
		FRBA	5.26	**17.29**	77.44		
		PICK3	10.45	**20.90**	68.66		
		Default	8.11	**16.22**	75.68		
	1.0	CACD	0.00	**1.15**	98.85	25.80	0.00
		FRBA	0.00	0.00	100.0		
		PICK3	0.00	**1.08**	98.92		
		Default	0.00	0.00	100.0		
BO All available S_{var} and S_{val}	0.05	CACD	23.11	**24.37**	52.52	25.60	0.81
		FRBA	20.73	**25.20**	54.07		
		PICK3	21.95	**26.02**	52.03		
		Default	19.23	**30.77**	50.00		
	0.1	CACD	23.85	**25.94**	50.21	16.19	0.80
		FRBA	**25.10**	21.86	53.04		
		PICK3	**25.91**	24.29	49.80		
		Default	**27.85**	24.05	48.10		
	0.2	CACD	**26.89**	21.85	51.26	13.00	1.21
		FRBA	**25.20**	19.92	54.88		
		PICK3	**26.83**	24.39	48.78		
		Default	**29.49**	25.64	44.87		
	0.5	CACD	**25.63**	23.95	50.42	8.82	1.56
		FRBA	**23.98**	21.14	54.88		
		PICK3	**26.02**	24.39	49.59		
		Default	**28.21**	24.36	47.44		
	1.0	CACD	2.94	**26.47**	70.59	8.53	2.40
		FRBA	3.25	**25.20**	71.54		
		PICK3	2.85	**26.42**	70.73		
		Default	2.56	**25.64**	71.79		

Table 4.1 Comprehensive results for all the possible ratios in comparison with the baselines.

Instance	5%		10%		20%		50%	
	Spearman	Kendall Tau	Spearman	Kendall Tau	Spearman	Kendall Tau	Spearman	Kendall Tau
CarpetCutting-test05	**0.94**	**0.83**	**0.96**	**0.92**	**0.91**	**0.84**	**0.89**	**0.81**
GeneralizedMKP-OR05x100-75-1	**0.99**	**0.99**	**0.99**	**0.99**	**0.92**	**0.84**	**0.91**	**0.80**
RIP-25-0-j120-01-01	**-0.33**	**-0.33**	**0.88**	**0.79**	**0.92**	**0.82**	**0.97**	**0.89**
KidneyExchange-4-081	**0.83**	**0.83**	**0.87**	**0.84**	**0.90**	**0.82**	**0.93**	**0.82**

Table 4.2 Average percentage of ranking correlation for the same percentage of the global timeout in 10 different runs — (5, 10, 20, 50)% of global timeout.

The correlation coefficient ranges from -1 to 1, where 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation. In our context, a high positive correlation suggests that the rankings obtained from a specific run with a specific timeout are similar to those obtained from another run with the same timeout.

This implies that if we run the approach with a fraction of the global timeout and it yields a ranking of the search strategies, the behavior of the search strategies tends to mirror the behavior observed with the same instance and with the same fraction of the global timeout but in a different run. This holds true even though the seed for the BO varies and PSA provides a different set and combination of search strategies each time. Therefore, if a superior search strategy is identified during a specific run, we can be reasonably confident that the same strategy would also be deemed superior when evaluated over another run.

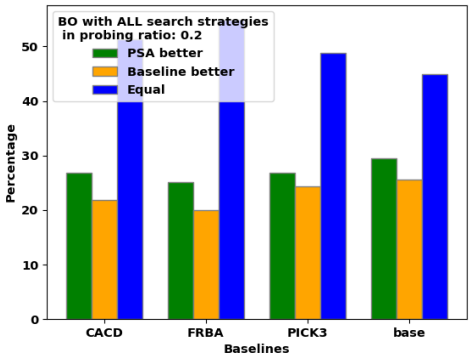
For instance, in *KidneyExchange-4-081*, the table shows an average correlation of 0.83 for all runs at first 5% of the global timeout to each other. This suggests that in all 10 runs of this specific instance, the behavior of search strategies in the first 5% of the global timeout were mostly the same as each other and they were correlated in 83% of the time.

The result of ranking correlations can be seen in Table 4.2.

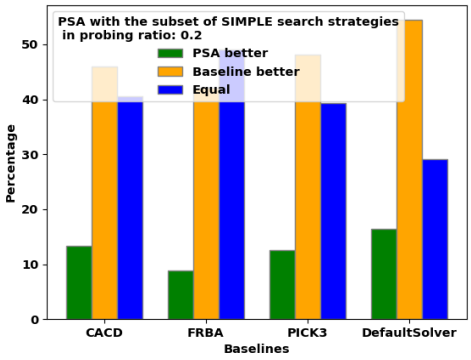
4.2.8.2 XCSP3 Benchmark with ACE Solver

In this subsection, we evaluate the performance of PSA on the XCSP3 problems. XCSP3 is an XML-based format designed to represent instances of combinatorial constraint problems from the perspective of CP. It is an intermediate integrated format that can represent each instance separately while preserving its structure [19].

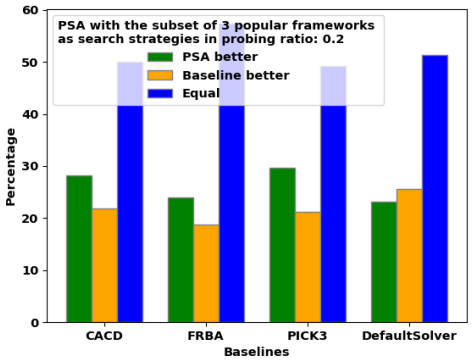
We compare the outcomes of PSA with four baselines. These baselines include three popular variable selection strategies: PICK3 (PickOnDom which is set with a pick degree of three and linked with variables when constraint propagation concludes with a conflict, as it was identified as the most effective degree [5]), DomWDeg/CACD, FrbaOnDom and the solver's default strategy. Performance is evaluated based on the objective value and the solving time. For the three popular variable selection strategies, the value selection strategy was left to the default of the solver as done in [5]. For the default search strategy of solvers, we do not



(a) All available S_{var} and S_{val} in comparison with the baselines.



(b) Simple set of S_{var} and S_{val} in comparison with the baselines.



(c) S_{var}=$\{PICK3, CACD, FRBA\}$ and all available S_{val} in comparison with the baselines.

Figure 4.2 Evaluation of PSA configurations across different variable–value strategy sets at a probing timeout ratio of 0.2

specify any variable or value selection strategy, leaving this decision to the solver.

First, to maintain consistency with the initial study, we evaluated the solver's performance by exploring the full set of variable and value selection strategies provided by ACE, as detailed in Section 4.2. This allows for a thorough evaluation of the effectiveness of different search strategies in improving the solver's performance.

Figure 4.1 compares the performance of PSA across three HPO methods—random search, Hyperband, and BO—at a probing timeout ratio of 0.2. Each subplot illustrates PSA's relative performance against four baseline strategies (CACD, FRBA, PICK3, and Default) across all available combinations of variable and value selection heuristics provided by the ACE solver. In each plot, the height of the bars represents the percentage of instances where PSA either surpassed, matched, or was surpassed by a given baseline, providing a clear comparison of how different HPO methods influence PSA's effectiveness under identical conditions.

In Fig. 4.2a, PSA performs better in 29.49% of the models than the default solver, and both yield equal results in 44.87% of the models, indicating that neither strategy consistently outperforms the other across all scenarios.

In the second step, we aim to evaluate the effectiveness of dynamic search strategies in comparison to static ones. Consequently, we select search strategies that are simpler to implement and do not require collecting statistics of propagators. The results obtained using a simple subset of variable and value selection strategies with a probing timeout ratio of 0.2, can be seen in Fig. 4.2b.

Variable and value selection strategies are shown below:

Variable selection strategies:

```
Deg, Rand, Lexico, Srand
```

Value selection strategies:

```
Vals, First, Last, RunRobin, Robin, Srand, Rand
```

The results indicate that PSA method surpasses the baseline methods in approximately 12% to 20% of the instances. However, the baseline methods demonstrate superior performance, outperforming PSA in around 50% of the instances, considering the fact that the baselines utilize dynamic methods to calculate and update weights. This could be interpreted as an indication that dynamic approaches surpass static approaches in this context. This superior performance of dynamic approaches could be attributed to their ability to adapt to changing conditions during the search process. Unlike static strategies, which use a fixed method throughout, dynamic strategies can adjust their methods based on the current state of the search. This adaptability allows them to navigate the search space more effectively, potentially avoiding local optima and finding better solutions.

Given that static search strategies appear to be less effective than dynamic ones, we employed our algorithm on a subset of three dynamic variable selection strategies. This was done to identify which subset of search strategies could yield enhanced performance. Consequently, we selected the three baselines, namely: PICK3, CACD, and FRBA as a subset of our variable selection strategies. The results of this approach with a probing timeout ratio of 0.2, can be observed in Fig. 4.2c.

In Fig. 4.2c, at a ratio of 0.2, PSA performed better in 23.98% of the models than FRBA, while this baseline performed better in 18.70% of the models. The results were equal in 57.32% of the models. This pattern is observed across different ratios and heuristics, with some variations, and PSA demonstrated a competitive performance against the baselines although PSA takes some time to probe.

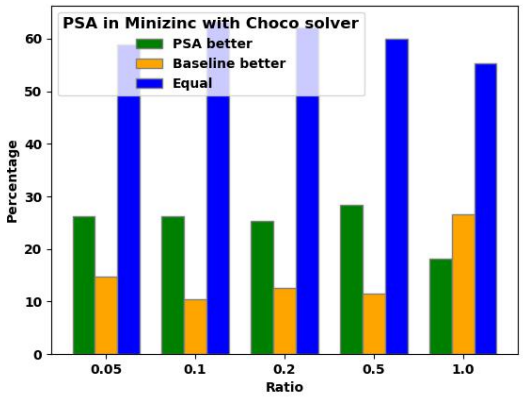


Figure 4.3 Performance comparison of PSA and the solver's default across different ratios in the MiniZinc framework with Choco solver.

4.2.8.3 MiniZinc Benchmark with Choco Solver

In this subsection, we concentrate on the efficacy of PSA when applied to the MiniZinc benchmark. MiniZinc is an open-source constraint modeling language. The MiniZinc models are subsequently compiled into FlatZinc, a solver input language comprehensible by a broad spectrum of solvers [99]. For this study, the utilized solver is Choco.

We compare the outcomes of our algorithm with the default search strategy, which is the combination of (`DomWDeg`, `Indomain_Min`) [101]. This combination is the preferred choice for the solver's default in the Choco solver.

The results of PSA on the MiniZinc benchmark problems, shown in Fig. 4.3, demonstrate its performance across a variety of problems. The results are more competitive than the solver's default strategy. When the ratio is set at 0.5, PSA outperforms the default search strategy, achieving better results in 28.42% of the instances. This is the highest percentage of instances where PSA performs better among all the ratios tested. Generally, PSA has demonstrated its potential by outperforming the default search strategy and baselines in certain scenarios.

The process of using MiniZinc involves compiling the models into FlatZinc [76] and then sending them to the solver. This process is inherently time-consuming as it involves both compilation and communication overheads. To address this, we have decided to compile the model to FlatZinc once and use it for the entire process of the experiment rather than in each run, significantly reducing the time and computational resources required. This change has contributed to the improved performance of PSA in the MiniZinc framework.

4.2.9 Detailed Analysis and PSA Performance

The comprehensive results derive from the various bar charts and tables are shown in Table 4.3 and Table 4.4. This includes a subset of variable and value selection strategies, both static and dynamic search strategies. These reveal that dynamic strategies tend to yield superior results. This can be attributed to several factors such as their ability to adapt to changing conditions, their capacity to learn from past decisions, and their potential to explore the search space more effectively.

We conducted an array of tests using different probing ratios, and the results indicate that a probing ratio of 20% generally outperforms the others for our specific set of problems. The results are presented in in Table 4.3 and Table 4.4, which uses the same format as the table in the previous phase, including the *Fallback to Default* and *Same Search Strategy* metrics to provide deeper insight into PSA's behavior.

4.2.10 Summary of Evaluation Findings

This evaluation with restricted hyperparameter spaces establishes that:

1. **Bayesian optimization** is the most effective HPO method for PSA, consistently outperforming random search and Hyperband.

2. A **20% probing ratio** provides optimal time allocation, balancing exploration during probing with sufficient solving time.

3. PSA demonstrates strong **robustness**, with statistical correlation confirming reliable strategy rankings from short probing times.

4. PSA **generalizes effectively** across different solvers (ACE, Choco) and modeling frameworks (XCSP3, MiniZinc).

5. The approach works best when given access to the solver's **full set of heuristics** rather than restricted subsets.

These findings validate PSA's core approach for search strategy selection and motivate evaluating its performance on complete solver configuration spaces.

Method		Baselines	Results (%)			Additional Results (%)	
			PSA Better	Baselines Better	Equal Results	Fallback to default	Same search strategy
XCSP3 with ACE solver All available S_{var} and S_{val}	0.05	CACD	23.11	**24.37**	52.52	25.60	0.81
		FRBA	20.73	**25.20**	54.07		
		PICK3	21.95	**26.02**	52.03		
		Default	19.23	**30.77**	50.00		
	0.1	CACD	23.85	**25.94**	50.21	16.19	0.80
		FRBA	**25.10**	21.86	53.04		
		PICK3	**25.91**	24.29	49.80		
		Default	**27.85**	24.05	48.10		
	0.2	CACD	**26.89**	21.85	51.26	13.0	1.21
		FRBA	**25.20**	19.92	54.88		
		PICK3	**26.83**	24.39	48.78		
		Default	**29.49**	25.64	44.87		
	0.5	CACD	**25.63**	23.95	50.42	8.82	1.56
		FRBA	**23.98**	21.14	54.88		
		PICK3	**26.02**	24.39	49.59		
		Default	**28.21**	24.36	47.44		
	1.0	CACD	2.94	**26.47**	70.59	8.53	2.40
		FRBA	3.25	**25.20**	71.54		
		PICK3	2.85	**26.42**	70.73		
		Default	2.56	**25.64**	71.79		
XCSP3 with ACE solver Simple set of S_{var} and S_{val}	0.05	CACD	13.08	**47.26**	39.66	28.16	0.00
		FRBA	11.02	**42.04**	46.94		
		PICK3	15.51	**47.35**	37.14		
		Default	14.29	**51.95**	33.77		
	0.1	CACD	15.90	**43.51**	40.59	23.88	0.00
		FRBA	11.34	**40.49**	48.18		
		PICK3	15.79	**44.53**	39.68		
		Default	16.46	**49.37**	34.18		
	0.2	CACD	13.39	**46.03**	40.59	18.62	0.00
		FRBA	8.91	**42.11**	48.99		
		PICK3	12.55	**48.18**	39.27		
		Default	16.46	**54.43**	29.11		
	0.5	CACD	12.61	**50.42**	36.97	16.26	0.00
		FRBA	9.35	**44.31**	46.34		
		PICK3	10.57	**53.25**	36.18		
		Default	17.95	**56.41**	25.64		
	1.0	CACD	0.00	**9.62**	90.38	15.38	0.00
		FRBA	0.00	**9.72**	90.28		
		PICK3	0.00	**9.72**	90.28		
		Default	0.00	**13.92**	86.08		

Table 4.3 Comprehensive evaluation of PSA with ACE solver across different probing ratios and strategy-set configurations (Part 1 of 2).

Method		Baselines	Results (%)			Additional Results (%)	
			PSA Better	Baselines Better	Equal Results	Fallback to default	Same search strategy
XCSP3 with ACE solver $S_{var} =$ PICK3, CACD, FRBA All available S_{val}	0.05	CACD	**25.52**	19.67	54.81	22.26	0.40
		FRBA	**23.89**	18.22	57.89		
		PICK3	**26.32**	23.08	50.61		
		Default	20.25	**27.85**	51.90		
	0.1	CACD	**27.31**	20.59	52.10	18.29	0.81
		FRBA	**27.24**	16.67	56.10		
		PICK3	**28.86**	22.76	48.37		
		Default	**19.23**	30.77	50.00		
	0.2	CACD	**28.15**	21.85	50.00	15.04	8.13
		FRBA	**23.98**	18.70	57.32		
		PICK3	**29.67**	21.14	49.19		
		Default	**23.08**	25.64	51.28		
	0.5	CACD	**25.52**	24.69	49.79	8.90	2.42
		FRBA	**23.89**	22.27	53.85		
		PICK3	**26.32**	25.10	48.58		
		Default	26.58	**27.85**	45.57		
	1.0	CACD	3.77	**28.03**	68.20	8.50	2.83
		FRBA	2.83	**26.72**	70.45		
		PICK3	2.83	**27.94**	69.23		
		Default	1.27	**30.38**	68.35		
MiniZinc with Choco solver	0.05	Default	**26.32**	14.74	58.95	36.84	0.00
	0.1	Default	**26.32**	10.53	63.16	31.58	1.05
	0.2	Default	**25.26**	12.63	62.11	22.11	0.00
	0.5	Default	**28.42**	11.58	60.00	23.68	1.05
	1.0	Default	18.09	**26.60**	55.32	18.30	1.06

Table 4.4 Comprehensive evaluation of PSA with ACE solver (dynamic subset) and Choco solver across different probing ratios (Part 2 of 2).

4.3 PSA on Complete Solver Configurations

4.3.1 Motivation and Experimental Goals

Building on the validated core approach, this evaluation assesses PSA's ability to handle the complete hyperparameter spaces of modern CP solvers. The goals are to:

- Systematically benchmark PSA's internal components (time management, timeout evolution, stopping conditions) to understand their individual and interactive effects.
- Identify optimal PSA configurations for different solver types and problem characteristics.
- Compare PSA's model-based approach against alternative tuning methods (Hamming distance baseline).
- Provide practical, data-driven guidance for configuring complex solvers under realistic time constraints.
- Demonstrate PSA's capability to handle the large configuration spaces of modern solvers.

4.3.2 Experimental Setup

This study employs an expanded experimental design covering the complete PSA architecture:

4.3.2.1 Solvers and Hyperparameter Spaces

To robustly evaluate PSA, we selected two distinct constraint solvers with varying complexities:

The ACE Solver ACE's extensive capabilities comply with the XCSP3-core standard, a widely recognized format for constraint problems [20],and offers a rich library of constraints, including advanced table constraints (ordinary, starred, and hybrid), and a wide array of global constraints such as `AllDifferent`, `Cardinality` [88], `Count`, `Element`, and `Cumulative` [69, 14, 9]. Additionally, it features various built-in search heuristics and is optimized for mono-criterion optimization. This comprehensive feature set is particularly valuable for PSA, as it contributes to a vast, complex, and highly influential hyperparameter space, making it an ideal candidate for showcasing the benefits of automated tuning.

The chosen hyperparameter space for ACE, summarized in Table 4.5, consists of 9 distinct dimensions that govern the solver's core search behavior. These include heuristics for variable and value selection (`Varh`, `Valh`), the information of table constraints (`sc2`, `sc3`), and other categorical parameters that guide branching, learning, and constraint handling. The combination of these chosen options results in a combinatorial search space of over 150 thousand unique configurations. This scale makes exhaustive tuning methods like grid search computationally infeasible and underscores the significant challenge of manual tuning. Consequently, ACE serves as an excellent testbed for demonstrating PSA's ability to effectively navigate such a hyperparameter space.

A practical contribution of this work is the integration of the previously unsupported ACE solver into the CPMpy library [23]. Within this newly integrated environment, we apply our HPO strategies to tune ACE across various problem instances. The framework's success is assessed based on both the quality of the final objective value obtained and the total time required to find that solution. Thus, utilizing ACE not only provides a challenging and relevant testbed for our tuning framework but also enhances the capabilities of the broader CPMpy ecosystem for future research.

The Choco Solver We also utilize the Choco solver (v4.10.14), a widely recognized open-source CP library implemented in Java. Choco provides a rich collection of heuristics and propagation levels, making it a strong alternative for benchmarking the PSA framework [84].

Parameter	Description	# Options
`Varh`	Variable Selection Heuristic	10
`Valh`	Value Selection Heuristic	10
`saf`	Stay Array Focus	2
`pc1`	Preserve Unary Constraints	2
`toh`	Convert to Hybrid Tables	2
`lc`	Last Conflict Weighting	4
`negative`	Algorithm for Negative Table Constraints	3
`sc2`	Structure for Binary Table Constraints	4
`sc3`	Structure for Ternary Table Constraints	4
Total Combinatorial Configurations		**153,600**

Table 4.5 The Tunable Hyperparameter Space of the ACE Solver.

Parameter	Description	# Options
Lc	Level of Consistency (Level of propagations)	2
Restarts	Restart policy (e.g., LUBY, GEOMETRIC, NONE)	12
Valh	Value Selection Heuristic	6
Varh	Variable Selection Heuristic	19
Flush	Propagation queue flush threshold	5
Total Combinatorial Configurations		**136,800**

Table 4.6 The Tunable Hyperparameter Space of the Choco Solver.

The hyperparameter space for Choco, which was derived from a JSON file detailing its available hyperparameters, is presented in Table 4.6. This space includes key components such as restart policies (`restarts`), variable and value selection heuristics (`Varh`, `Valh`), consistency levels (`lc`), and propagation queue flushing thresholds (`flush`).

4.3.2.2 Problem Instances

To ensure a thorough evaluation, our study utilized a set of 114 diverse problem instances, all formatted in the XCSP3 XML standard. These instances were carefully selected to represent a wide range of combinatorial challenges. All experiments were conducted under strictly controlled conditions to guarantee fairness and reproducibility, using the HPC facilities previously described in Section 4.1.1. A global wall-clock time limit of 1800 seconds (30 minutes) was allocated for each individual run, consistent with standard durations used in the XCSP3 competition. Where random seeds were applicable, the same seed was consistently used across all comparable runs to maintain fairness and ensure reproducible results.

4.3.2.3 Benchmarking Methodology

Our benchmarking approach involved four distinct types of experimental runs for both the ACE and Choco solvers:

Default Solver Baselines For each of the 114 instances, we first solve the instances with the default version of both ACE and Choco solvers. These runs served as a foundational baseline, reflecting the out-of-the-box performance of each solver without any hyperparameter tuning. Each default run adhered to the same system environment and the 1800-second global time limit.

PSA Framework Configurations The core of our extensive benchmarking involves the PSA framework, which employs two distinct HPO methods: BO and Hamming distance search [16]. Both methods are implemented within the PSA framework, allowing a direct comparison of their effectiveness. We conduct a full-factorial experiment to systematically evaluate the impact of each modular component of the PSA framework. This design allows us to not only identify the best overall configuration but also to analyze the independent contribution of each strategic dimension to performance. A total of 24 unique PSA configurations for each HPO method are generated by combining all possibilities from the following strategic dimensions:

- **Probing Time Allocation:** 2 options (*Static*, *Dynamic*)
- **Round Timeout Initialization:** 2 options (*Static*, *First-Runtime*)
- **Timeout Evolution Pattern:** 3 options (*Static*, *Geometric*, *Luby*)
- **Probing Stop Condition:** 3 options (*First Solution*, *Timeout*, *Stagnation*)

Some combinations of these settings are inherently incompatible. In particular, when the probing stop condition is set to *First Solution*, the probing phase terminates as soon as the first solution is found. In this case, the probing loop executes only a single round, which makes any form of round-timeout evolution irrelevant. For this reason, whenever the stop condition is *First Solution*, only the *static* timeout evolution pattern is meaningful and permitted. Accordingly, these 24 configurations are derived from two batches:

- Round-timeout static: $2 \times 3 \times 3 = 18$ configurations
- Round-timeout First-Runtime: $2 \times 1 \times 3 = 6$ configurations

These 24 PSA configurations for each HPO method are evaluated per instance for both the ACE and Choco solvers.

Champion Configuration Evaluation Following the exploratory phase of the PSA framework, the single best-performing configuration (the *champion configuration*) for each combination of solver and HPO method is identified based on the aggregated results. Subsequently, these identified champion configurations are then run separately on all 114 instances. These dedicated runs provide a direct and robust measure of their performance for comparative analysis against the baselines.

4.3.2.4 Execution Environment and Data Collection

To ensure fairness and reproducibility, all experiments were conducted under strictly controlled conditions.

- **Global time limit:** Each individual experimental run is allocated a maximum wall-clock time of 1800 seconds (30 minutes), aligning with the standard duration used in the XCSP3 competition from which our benchmarks are drawn.
- **Total computational effort:** The complete experiment, encompassing 6156 individual runs (114 instances $\times$ (24 PSA configurations $\times$ 2 HPO methods + 1 default solver baseline configuration + 1 PSA champion for each HPO method) $\times$ 2 solvers), represents a total of 3078 compute-hours.
- **Data logging and reproducibility:** For each run, detailed performance data is logged to structured CSV files:
  - **Objective value:** The best objective value found by the solver within the time limit.
  - **Solver status:** The final exit status reported by the solver (e.g., `OPTIMUM FOUND`, `SATISFIABLE`, `TIMEOUT`, `ERROR`).
  - **Runtime:** The total wall-clock time elapsed for the run.
  - **Configuration details:** All command-line flags used for each run are logged to ensure full reproducibility and allow detailed performance analysis.
- **Parameter Settings for PSA Components:** For strategies utilizing the *Static Percentage Allocation*, the probing budget was set to 20% of *GT*, a choice that tries to balance exploration with a substantial budget for the final solve. The *Static* initial round timeout was set to a baseline of *5* seconds. For the *Geometric* evolution pattern, a growth factor of $\beta = 1.5$ is used. These values are chosen as reasonable, standard defaults to avoid overfitting the framework to this specific benchmark set.

Component	Strategy	BO Wins	Hamming Wins
Global Time	Percent	53.40%	42.98%
	Dynamic	46.60%	57.02%
Round Timeout	Static	76.70%	81.58%
	First-Runtime	23.30%	18.42%
Timeout Evolution	DynamicGeometric	31.07%	28.07%
	DynamicLuby	31.07%	20.18%
	Static	37.86%	51.75%
Stop Condition	Timeout	36.89%	38.6%
	Stagnation	35.92%	27.19%
	FirstSolution	27.18%	34.21%

Table 4.7 PSA Component Strategy Frequencies for Choco Solver

4.3.3 Results and Analysis

This section presents the comprehensive results of our large-scale benchmarking experiments. Our primary objective is to identify the most effective automated strategies for configuring a solver's hyperparameters within the PSA framework and to derive key principles for efficiently utilizing a limited time budget. We organize our findings into three main stages: for each solver, starting with Choco and then proceeding to ACE, first, we provide an aggregate analysis of the performance of PSA's internal component strategies, examining general trends across all problem instances. Second, based on these aggregate insights, we synthesize and present the champion configurations. Third, we then conduct detailed solver-specific performance comparisons, evaluating how these identified champion configurations perform against baseline configurations. For clarity and focus, the numerical results and pie charts presented in this section derive from a single, representative random seed.

We present the aggregated results in four comprehensive tables. Table 4.7 shows the frequency of winning PSA component strategies for the Choco solver with both BO and Hamming HPO methods. Table 4.8 provides the equivalent analysis for ACE. Based on these frequencies, we derive champion configurations for each solver-HPO combination in Table 4.9. Finally, Table 4.10 summarizes the performance comparisons between PSA-tuned and default configurations.

4.3.3.1 Choco Solver Results

Analysis of PSA Component Strategies for Choco Table 4.7 reveals distinct patterns in effective PSA strategies for the Choco solver. For global time management, BO favors the

Component	Strategy	BO Wins	Hamming Wins
Global Time	Percent	50.44%	48.25%
	Dynamic	49.56%	51.75%
Round Timeout	Static	77.18%	86.84%
	First-Runtime	22.82%	13.16%
Timeout Evolution	DynamicGeometric	48.73%	27.19%
	DynamicLuby	25.17%	49.12%
	Static	26.10%	23.68%
Stop Condition	Timeout	35.59%	29.82%
	Stagnation	33.94%	35.96%
	FirstSolution	30.46%	34.21%

Table 4.8 PSA Component Strategy Frequencies for ACE Solver

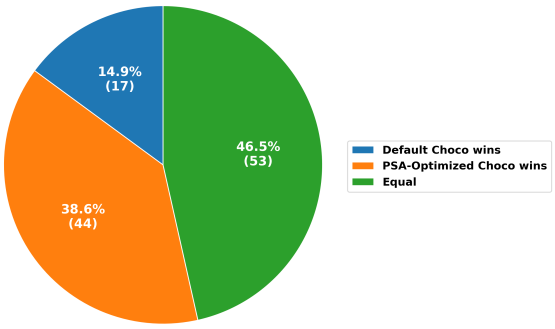
Component	Choco Champion BO / Hamming	ACE Champion BO / Hamming
Global Time Management	Percent / Dynamic	Percent / Dynamic
Round Timeout Initialization	Static / Static	Static / Static
Timeout Evolution Pattern	DynamicGeometric / Static	DynamicGeometric / DynamicLuby
Probing Stop Condition	Timeout / Timeout	Timeout / Stagnation

Table 4.9 Champion Configurations for Both Solvers

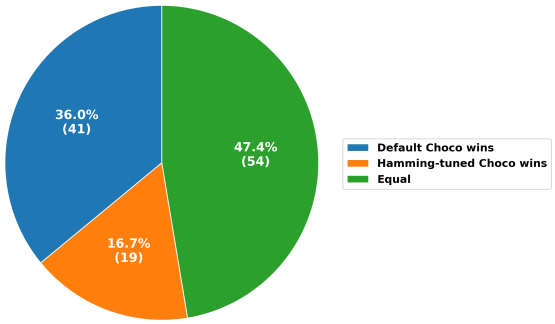
percent strategy (53.40%), while Hamming prefers *dynamic* allocation (57.02%). Both HPO methods strongly favor *static* round timeout initialization (76.70% for BO, 81.58% for Hamming). Timeout evolution patterns differ significantly: with BO, all three strategies are nearly equally effective (37.86% static, 31.07% each for dynamic variants), whereas with Hamming, *static* evolution dominates (51.75%). For stop conditions, *timeout* is most frequent with both methods, though Hamming shows stronger preference for *firstSolution* (34.21%) compared to BO (27.18%).

Synthesizing Champion Configurations for Choco Based on the aggregated analysis in Table 4.7, we derive two champion configurations for Choco, one optimized for BO and another for Hamming. These configurations balance component effectiveness with overall strategy coherence and are summarized in Table 4.9. For BO, the champion combines *percent* global time, *static* timeout initialization, *dynamicGeometric* evolution, and *timeout* stop condition. For Hamming, the configuration uses *dynamic* global time, *static* timeout initialization, *static* evolution, and *timeout* stop condition.

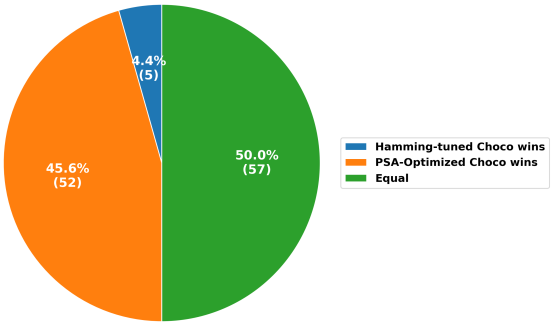
Choco Performance Comparisons To evaluate PSA's effectiveness for Choco, we compare three approaches: default settings (*default Choco*), PSA with BO (*PSA-BO Choco*), and PSA



(a) PSA-BO Choco vs. Default Choco



(b) PSA-Hamming Choco vs. Default Choco



(c) PSA-BO Choco vs. PSA-Hamming Choco

Figure 4.4 Pairwise Performance Comparison for Choco Solver Approaches. (a) illustrates the distribution of outcomes for *PSA-BO Choco* versus *default Choco*. (b) shows the comparison between *PSA-Hamming Choco* and *default Choco*. (c) details the head-to-head results of *PSA-BO Choco* against *PSA-Hamming Choco*.

with Hamming (*PSA-Hamming Choco*). Results are summarized in Table 4.10 and visualized in Figure 4.4.

PSA-BO Choco vs. Default Choco: PSA-BO Choco performs better on 38.6% of instances, while default Choco is superior on only 14.9%, with ties on 46.5% (Figure 4.4a). This demonstrates BO's advantage over default settings.

PSA-Hamming Choco vs. Default Choco: Default Choco outperforms PSA-Hamming on 24.6% of instances, while PSA-Hamming wins on 29.8%, with ties on 45.6% (Figure 4.4b). This indicates PSA Hamming-based tuning generally outperforms Choco's default configurations.

PSA-BO Choco vs. PSA-Hamming Choco: PSA-BO clearly outperforms PSA-Hamming, winning on 23.7% of instances versus only 14.9% for Hamming, with ties on 61.4% (Figure 4.4c). This validates BO's superiority over the simpler Hamming approach.

4.3.3.2 ACE Solver Results

Analysis of PSA Component Strategies for ACE Table 4.8 shows PSA strategy effectiveness for ACE. Global time management shows minimal preference: BO slightly favors

Comparison	Winner A	Tie	Winner B
Choco Results			
PSA-BO Choco vs Default	38.6%	46.5%	14.9%
PSA-Hamming Choco vs Default	29.8%	45.6%	24.6%
PSA-BO vs PSA-Hamming Choco	23.7%	61.4%	14.9%
ACE Results			
PSA-BO ACE vs Default	25.4%	57.9%	16.7%
PSA-Hamming ACE vs Default	13.2%	68.4%	18.4%
PSA-BO vs PSA-Hamming ACE	24.6%	64.9%	10.5%

Table 4.10 Performance Comparison Summary (Percentage of Instances)

percent (50.44%), Hamming slightly favors *dynamic* (51.75%). Both strongly prefer *static* timeout initialization (77.18% BO, 86.84% Hamming). Timeout evolution differs dramatically: BO strongly prefers *dynamicGeometric* (48.73%), while Hamming favors *dynamicLuby* (49.12%). Stop conditions show balanced distributions, with *timeout* most frequent for BO (35.59%) and *stagnation* for Hamming (35.96%).

Synthesizing Champion Configurations for ACE Based on Table 4.8, we derive champion configurations for ACE (Table 4.9). For BO: *percent* global time, *static* timeout initialization, *dynamicGeometric* evolution, and *timeout* stop condition. For Hamming: *dynamic* global time, *static* timeout initialization, *dynamicLuby* evolution, and *stagnation* stop condition.

Performance Comparison of ACE Configurations We compare default ACE with PSA-BO ACE and PSA-Hamming ACE. Results are in Table 4.10 and Figure 4.5.

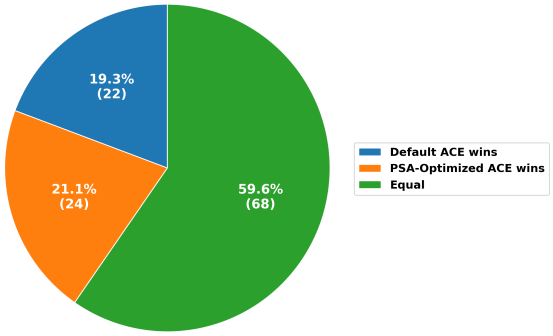
PSA-BO ACE vs. Default ACE: PSA-BO ACE wins on 25.4% of instances versus 16.7% for default, with ties on 57.9% (Figure 4.5a). BO provides clear advantage over default settings.

PSA-Hamming ACE vs. Default ACE: Default ACE outperforms PSA-Hamming on 18.4% of instances, while PSA-Hamming wins on 13.2%, with ties on 68.4% (Figure 4.5b). Hamming-based tuning generally underperforms defaults.

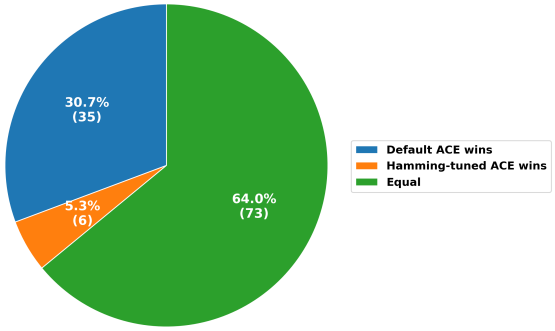
PSA-BO ACE vs. PSA-Hamming ACE: PSA-BO dominates, winning on 24.6% of instances versus 10.5% for Hamming, with ties on 64.9% (Figure 4.5c). BO's model-based approach proves significantly more effective.

4.3.4 Summary of Complete Configuration Evaluation Findings

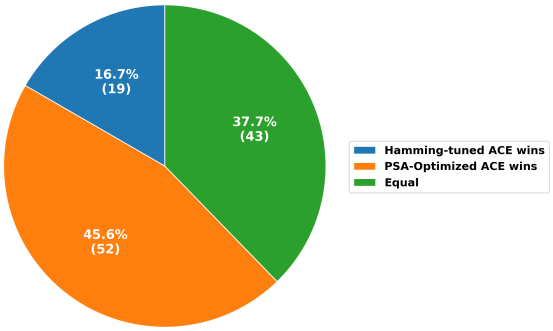
This evaluation with complete solver configurations reveals several key insights:



(a) PSA-BO ACE vs. Default ACE



(b) PSA-Hamming ACE vs. Default ACE



(c) PSA-BO ACE vs. PSA-Hamming ACE

Figure 4.5 Pairwise Performance Comparison for ACE Solver Approaches. (a) illustrates the distribution of outcomes for *PSA-BO ACE* versus *default ACE*. (b) shows the comparison between *PSA-Hamming ACE* and *default ACE*. (c) details the head-to-head results of *PSA-BO ACE* against *PSA-Hamming ACE*.

1. PSA effectively handles **large configuration spaces**, demonstrating its scalability to real-world solver tuning.

2. **Component interactions matter**—optimal configurations differ between solvers and HPO methods, emphasizing the need for solver-specific tuning.

3. **Dynamic timeout evolution** (particularly geometric for BO, Luby for Hamming) generally outperforms static approaches, enabling adaptive resource allocation.

4. PSA with BO **significantly outperforms** both default configurations and Hamming-based tuning, validating the value of model-based optimization.

5. Optimal configurations show **similar patterns** across solvers (static timeout initialization, timeout stop condition), suggesting general principles for solver tuning.

6. The **20% probing ratio** remains effective even for complete configuration spaces, demonstrating consistency across different problem complexities.

4.4 Discussion and Synthesis

4.4.1 Unifying Insights from Complementary Evaluations

The two-tiered experimental design, beginning with restricted hyperparameter spaces and advancing to complete solver configurations, has yielded a cohesive understanding of PSA's capabilities. This progression reveals not only the framework's immediate effectiveness but also its fundamental properties and scaling behavior.

Scalability Demonstrated through Progressive Complexity. Starting with the focused domain of search strategy selection, we established PSA's core competency: efficiently identifying effective branching heuristics through Bayesian optimization with a fixed 20% time allocation. This foundational result served as a springboard for the more ambitious evaluation on complete solver configurations. The successful extension to these complex spaces—exemplified by ACE's 153,600-parameter landscape, confirms that PSA's design principles scale gracefully. The framework maintains its effectiveness even when navigating the intricate interdependencies present in modern solver architectures, transitioning seamlessly from specialized component tuning to holistic solver optimization.

Consistent Performance Patterns Across Experimental Contexts. Three findings emerged as remarkably stable across both evaluation tiers. First, Bayesian optimization consistently proved superior to simpler alternatives, validating the value of model-based exploration in constrained time scenarios. Second, the 20% probing ratio demonstrated robust optimality, suggesting this allocation effectively balances the exploration–exploitation trade-off inherent in time-bounded configuration. Third, PSA exhibited impressive generalization, delivering competitive results across diverse solvers (ACE and Choco) and modeling paradigms (XCSP3 and MiniZinc). These consistent outcomes suggest that PSA captures fundamental principles of efficient solver configuration rather than exploiting niche characteristics of specific test environments.

4.4.2 Theoretical Contributions and Practical Significance

Our experimental findings contribute to both the theoretical understanding of automated solver configuration and provide actionable insights for practitioners.

Advancing Algorithm Configuration Theory. The empirical validation of PSA's two-phase architecture offers concrete evidence for the probe-solve paradigm's efficacy in time-constrained optimization scenarios. By cleanly separating exploration from exploitation, PSA provides a template for resource-aware algorithm design. Furthermore, the consistent superi-

ority of Bayesian optimization over simpler methods (random search, Hyperband, Hamming distance) reinforces the theoretical premise that intelligent, model-based search provides substantial advantages in complex configuration spaces. The effectiveness of dynamic timeout strategies, particularly geometric progression, adds nuance to our understanding of how to allocate evaluation resources adaptively during the exploration phase.

Empowering Practical Solver Deployment. For constraint programming practitioners, PSA offers several immediate benefits. The framework operates in a nearly parameterless fashion—users need only specify the total time budget—lowering the barrier to automated configuration. The champion configurations identified for ACE and Choco provide practical starting points for users of these solvers. Beyond these specific recommendations, the consistent patterns observed (static initialization followed by adaptive exploration) suggest general heuristics applicable to other CP solvers. The integration of ACE into the CPMpy ecosystem represents a tangible contribution that extends the toolbox available to the research community. Perhaps most importantly, PSA demonstrates that sophisticated configuration need not be reserved for experts; it can be automated effectively within realistic computational constraints.

4.4.3 Limitations and Future Research Directions

While comprehensive within its scope, this study naturally suggests avenues for further investigation and refinement of the PSA framework.

Expanding the Empirical Scope. Our evaluation focused on XCSP3 and MiniZinc benchmarks using two Java-based solvers. Future work could assess PSA's effectiveness across a broader ecosystem, including additional solvers (such as OR-Tools, Gecode, or CP-SAT) and problem domains beyond those represented in standard competition benchmarks. Such expansion would further test the framework's generalizability and potentially reveal domain-specific adaptation requirements.

Enhancing Framework Adaptivity. Several dimensions of PSA could benefit from increased intelligence. The currently fixed 20% probing ratio, while effective, might be optimized dynamically based on problem characteristics or runtime observations. The framework could also be extended to handle multi-objective optimization, balancing solution quality against runtime or memory consumption. Additionally, parallelizing the probing phase could significantly increase configuration efficiency for high-performance computing environments.

Toward Predictive Configuration. A particularly promising direction involves moving beyond purely empirical configuration toward predictive approaches. By analyzing instance features (such as constraint graph properties, variable-to-constraint ratios, or domain sizes),

future versions of PSA might predict promising configurations without extensive probing, or at least intelligently bias the initial search. Such feature-based approaches could dramatically reduce configuration overhead for recurring problem types.

4.4.4 Concluding Synthesis

This chapter has systematically validated the PSA framework through methodical experimentation at two levels of complexity. The restricted evaluation established foundational principles: Bayesian optimization's superiority, the 20% time allocation heuristic, and PSA's cross-solver robustness. The complete configuration evaluation demonstrated these principles scale effectively to handle realistic solver complexity, with PSA consistently outperforming default configurations and simpler tuning methods.

The synthesis of these findings reveals PSA as more than a collection of effective techniques; it represents a coherent approach to time-bounded algorithm configuration. By intelligently partitioning the time budget, employing model-based search, and adaptively managing evaluation resources, PSA addresses the practical challenge of solver tuning under constraints that mirror real-world usage.

Seven key conclusions emerge from this comprehensive evaluation:

1. **Bayesian optimization supremacy:** Model-based search significantly outperforms simpler alternatives for constrained-time configuration.

2. **Optimal time partitioning:** A 20% probing allocation balances exploration and exploitation across diverse problems.

3. **Cross-platform generalization:** PSA's effectiveness extends across solver architectures and modeling frameworks.

4. **Adaptive resource management:** Dynamic timeout strategies (particularly geometric progression) enhance configuration quality.

5. **Superiority over baselines:** PSA with BO consistently outperforms both default configurations and simpler tuning methods.

6. **Scalability to complexity:** The framework handles configuration spaces exceeding 150,000 parameters effectively.

7. **Practical design pattern:** Optimal configurations consistently combine static initialization with adaptive exploration.

Collectively, these results establish PSA as a robust, scalable approach to constraint solver configuration. By making sophisticated hyperparameter optimization accessible under realistic time constraints, PSA advances both the theory and practice of automated algorithm design for constraint programming. The framework provides not only immediate utility for practitioners but also a foundation for future research into increasingly intelligent and adaptive configuration methods.

4.4.5 Published Work

The experimental methodology and findings presented in this chapter have been disseminated through several academic venues:

- HADDAD, H., TALBOT, P., & BOUVRY, P. (10 July 2024). *Comparison of Hyperparameter Optimization Methods for Selecting Search Strategy of Constraint Programming Solvers* [Paper presentation]. PTHG-24: The Seventh Workshop on Progress Towards the Holy Grail, Girona, Spain. URL: https://hdl.handle.net/10993/63582
- H. Haddad, P. Talbot and P. Bouvry, "Selecting Search Strategy in Constraint Solvers using BO," *2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI)*, Herndon, VA, USA, 2024, pp. 764-773, doi: 10.1109/ICTAI62512.2024.00113.

Part III

Domain Intelligence: Real-World Applications in Construction

Chapter 5

Standard-Based BIM Optimization in the Construction Domain

Contents

5.1 Introduction

This chapter presents preliminary and complementary work on the role of BIM, AI, and CP in construction-oriented optimization. It serves as a conceptual bridge between general BIM-

driven decision-support techniques and the more specialized, standards-based CP framework developed later in Chapter 6.

Two themes are explored:

- The first part reviews how BIM supports construction digitalization, how it interacts with AI and multi-objective optimization, and which technical challenges arise when such systems must be accurate, explainable, and trustworthy [90, 64].
- The second part demonstrates how a construction standard such as ISO 10077-1:2017 on thermal transmittance can be encoded as a formal constraint model. This provides a concrete example of how domain knowledge traditionally expressed in textual standards can be transformed into computable constraints to support automated reasoning [80].

Together, these components illustrate the progression from high-level BIM-based optimization concepts to explicit, standard-driven CP models. This progression also motivates the development of the full BIM-to-CP pipeline introduced in Chapter 6, where engineering standards are integrated into multi-objective optimization of real building elements [104].

5.2 Building Information Modeling, AI, and Trustworthiness

5.2.1 Automation in Construction and Building Information Modeling

Prior to the fourth industrial revolution, construction optimization was constrained by existing manual methods. Project management relied heavily on human expertise and trial-and-error strategies [57].

The increased adoption of automation, AI, internet of things (IoT), and other technological breakthroughs associated with the fourth industrial revolution has progressively transformed the construction sector into a more digitized industry. In particular, this evolution has accelerated the development and deployment of BIM [90].

5.2.1.1 Building Information Modeling

BIM is a digital representation of a building's physical and functional characteristics, potentially spanning the entire lifecycle, from early design to construction, renovation, operation, and eventual demolition [90]. Modern BIM models are increasingly data-driven, collaborative, and

predictive [63]. They reduce error-prone manual data entry, enable early detection of design flaws, and support informed decision-making through integrated simulations [40].

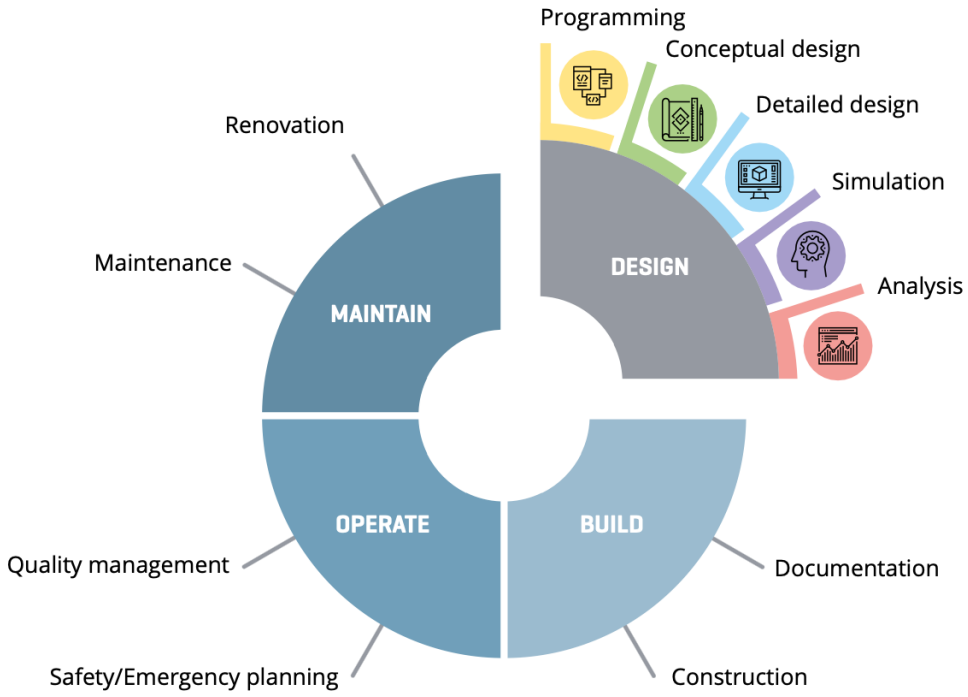


Figure 5.1 Role of BIM within a building design and construction workflow.

Examples of technical aspects of a construction project that BIM can support include:

- forecasting and optimizing energy consumption during operation;
- predicting and optimizing daylight usage [104];
- optimizing indoor air quality;
- enhancing thermal comfort for building occupants [75];
- improving aesthetic considerations such as access to views and natural lighting;
- optimizing construction scheduling, sequencing, and duration;
- selecting sustainable building materials and minimizing waste.

5.2.1.2 The Relation Between BIM and Modern ICT Tools

BIM can be coupled with various modern technologies, particularly AI-based methods, to produce dynamic simulations and facilitate multi-objective design decisions [63]. MOO algorithms are natural candidates for identifying configurations that satisfy competing criteria [26, 107].

MOO is connected to a broad range of techniques, including genetic algorithms, particle swarm optimization, ant colony optimization, simulated annealing, and hybrid AI-optimization approaches. ML further enhances BIM-based analysis by predicting performance outcomes from historical data [8].

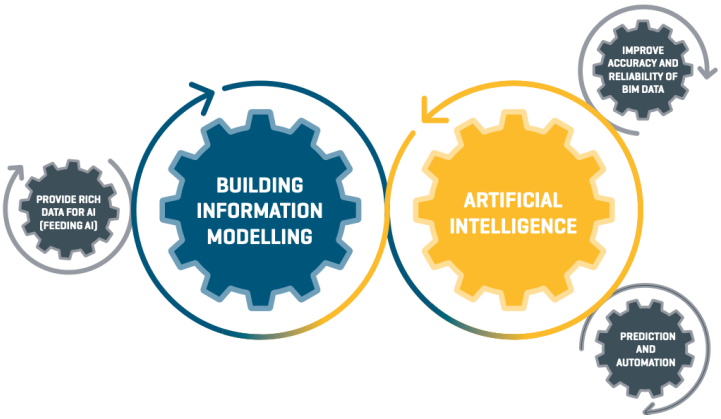


Figure 5.2 Conceptual representation of interactions between BIM and AI.

In such interactions, AI can:

- improve the accuracy and completeness of BIM data (through inference, anomaly detection, etc.);
- use BIM as structured input for predictive modeling;
- automate decision support for design and operation [40].

5.2.1.3 Trustworthiness Issues in BIM

Given the potential consequences of construction errors—ranging from minor aesthetic issues to life-threatening structural flaws—trustworthiness is critical in BIM-supported workflows [85].

Relevant trustworthiness characteristics include:

- **Accuracy:** ensuring that digital designs reflect real-world behavior.
- **Accountability:** especially when AI components influence decisions.
- **Quality:** requiring vast amounts of accurate, complete, and consistent data [90].
- **Transparency:** enabling stakeholders to validate models and data.
- **Usability and Interoperability:** ensuring that BIM data is accessible and consistent across software tools.

5.2.1.4 BIM-based Multi-Objective Optimization

The research aims to produce a near-optimal window design that simultaneously minimizes construction cost and energy consumption while maximizing daylighting [104]. The optimization framework integrates variables such as building orientation, number and size of windows, and material choices. To structure this methodology and its connection to standardization, the workflow is decomposed into: (i) input data, (ii) the system, and (iii) the output.

Input Data and Information Construction standards provide a reliable foundation for defining input data [80]. Adhering to standardized formats ensures consistency, accuracy, and interoperability across projects. Representative standards used as input are listed in Table 5.1.

Further enhancing the model's accuracy, the integration of real-world environmental and operational data is crucial. This includes actual energy consumption patterns, prevailing environmental conditions, and occupant behavior. Incorporating such data enables more realistic performance predictions and ensures greater compatibility with real building contexts [75]. Relevant standards guiding these data sources are presented in Table 5.1.

Input Software The architectural model central to this research was developed using Autodesk Revit, a widely adopted BIM authoring platform [90]. Dynamo, a visual programming extension for Revit, was used to automate the generation and configuration of window geometry and orientation parameters, which are essential for optimization [40].

Standardization also extends to software quality, interoperability, and testing processes. These aspects contribute significantly to trustworthiness in BIM-based workflows [85]. Representative standards are listed in Table 5.4.

[1] https://www.iso.org/standard/68078.html
[2] https://www.iso.org/committee/49180.html
[3] https://www.iso.org/standard/75401.html
[4] https://www.iso.org/standard/70303.html
[5] https://www.iso.org/standard/46394.html
[6] https://www.iso.org/committee/4418564.html
[7] https://www.iso.org/standard/67090.html
[8] https://www.iso.org/committee/53512.html
[9] https://www.iso.org/standard/65601.html
[10] https://www.iso.org/committee/53476.html
[11] https://www.iso.org/standard/67002.html
[12] https://www.iso.org/standard/38608.html
[13] https://www.iso.org/committee/55238.html
[14] https://www.iso.org/standard/55154.html
[15] https://www.iso.org/standard/45086.html
[16] https://www.iso.org/standard/64764.html

Standard title	Standardization committee	Scope extract / description	Trustworthiness characteristics affected
ISO 19650-1:2018[1] **Organization and digitization of information about buildings and civil engineering works, including building information modeling (BIM) — Information management using building information modeling — Part 1: Concepts and principles**	ISO/TC 59/SC 13 Organization and digitization of information about buildings and civil engineering works, including building information modeling (BIM)[2]	This document outlines the concepts and principles for information management at a stage of maturity described as "building information modeling (BIM) according to the ISO 19650 series". This document provides recommendations for a framework to manage information including exchanging, recording, versioning and organizing for all actors. This document is applicable to the whole life cycle of any built asset, including strategic planning, initial design, engineering, development, documentation and construction, day-to-day operation, maintenance, refurbishment, repair and end-of-life.	Quality, Accuracy
	Relation to research topic: The ISO 19650 series in general (composed of 5 parts) is on good practices for information management across all stakeholders in a construction project through BIM. It warrants a need for interoperability and adequate cooperation between these stakeholders and the underlying systems they use. Input data following such principles ultimately benefit the overall system. The BIM data and model could be better optimized using these principles		
ISO 23386:2020[3] **Building information modeling and other digital processes used in construction — Methodology to describe, author and maintain properties in interconnected data dictionaries**	ISO/TC 59/SC 13 Organization and digitization of information about buildings and civil engineering works, including building information modeling (BIM)	This document establishes the rules for defining properties used in construction and a methodology for authoring and maintaining them, for a confident and seamless digital share among stakeholders following a BIM process.	Quality, Usability, Transparency
	Relation to research topic: Interoperable and commonly accepted data formats are necessary in order to be able to optimize multiple dimensions, in particular daylight optimization, energy performance, and cost calculation, and keep the methods transparent for stakeholders.		
ISO 16739-1:2018[4] **Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries — Part 1: Data schema**	ISO/TC 59/SC 13 Organization and digitization of information about buildings and civil engineering works, including building information modeling (BIM)	The Industry Foundation Classes, IFC, are an open international standard for Building Information Model (BIM) data that are exchanged and shared among software applications used by the various participants in the construction or facility management industry sector. The standard includes definitions that cover data required for buildings over their life cycle. The Industry Foundation Classes specify a data schema and an exchange file format structure.	Quality, Usability, Transparency
	Relation to research topic: Interoperable and commonly accepted data formats are necessary in order to be able to optimize multiple dimensions, in particular daylight optimization, energy performance, and cost calculation, and keep the methods transparent for stakeholders.		

Table 5.1 Standards potentially useful as input to the system (Part 1).

Standard title	Standardization committee	Scope extract / description	Trustworthiness characteristics affected
ISO 10916:2014[5] Calculation of the impact of daylight utilization on the net and final energy demand for lighting	ISO/TC 274 Light and lighting [6]	ISO 10916:2014 defines the calculation methodology for determining the monthly and annual amount of usable daylight penetrating non-residential buildings through vertical facades and rooflights and the impact thereof on the energy demand for electric lighting. It can be used for existing buildings and the design of new and renovated buildings.	Accuracy
	Relation to research topic: Standardized calculation methodologies for lighting estimations and the resulting electrical demand for artificial lighting can be integrated to the formula to optimize when weighing natural light against energy consumption in BIM models. Thus, this serves the dual purpose of daylight optimization and energy performance.		
ISO 10077-1:2017[7] Thermal performance of windows, doors and shutters, Calculation of thermal transmittance	ISO/TC 163/SC 2 Calculation methods [8]	ISO 10077-1:2017 specifies methods for the calculation of the thermal transmittance of windows and pedestrian doors consisting of glazed and/or opaque panels fitted in a frame, with and without shutters.	Accuracy
	Relation to research topic: Also related to the energy optimization of daylight usage is the effect of natural lighting generating heat through translucent and transparent material. Standardized formula to estimate this can be used in BIM models.		

Table 5.2 Standards potentially useful as input to the system (Part 2).

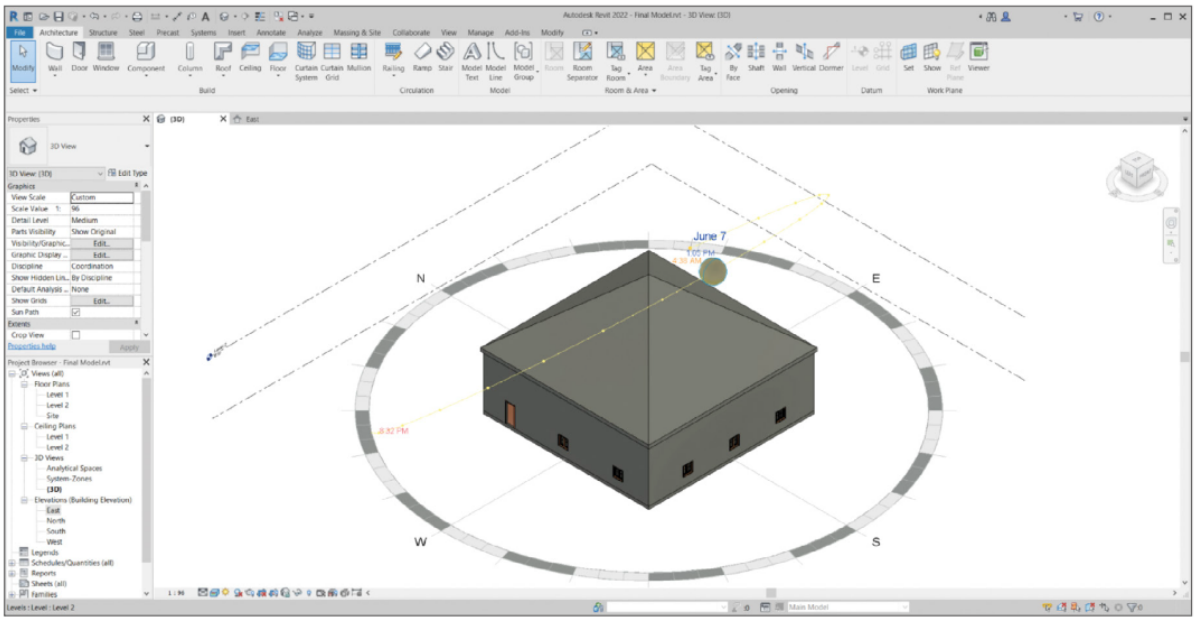


Figure 5.3 Representative building model developed in Autodesk Revit, illustrating building orientation, solar path, and parameterized window placements.

Standard title	Standardization committee	Scope extract / description	Trustworthiness characteristics affected
ISO 52000-1:2017[9] **Energy performance of buildings — Overarching EPB assessment — Part 1: General framework and procedures**	ISO/TC 163 [10]	ISO 52000-1:2017 establishes a systematic, comprehensive and modular structure for assessing the energy performance of new and existing buildings (EPB) in a holistic way. It is applicable to the assessment of overall energy use of a building, by measurement or calculation, and the calculation of energy performance in terms of primary energy or other energy-related metrics. It takes into account the specific possibilities and limitations for the different applications, such as building design, new buildings 'as built', and existing buildings in the use phase as well as renovation.	Reliability, Accuracy, Accountability
	Relation to research topic: Importance is attached to making realistic predictions on energy consumption in a given model, or it will have only limited use. Thus, having a standardized methodology that can estimate energy consumption after the fact in order to have a realistic baseline for comparison with the model predictions is extremely useful, in particular to keep the underlying system – which may be AI-supported – accountable for its predictions.		
ISO/CIE 20086:2019[11] **Light and lighting — Energy performance of lighting in buildings**	ISO/TC 274 Light and lighting	This document specifies the methodology for evaluating the energy performance of lighting systems for providing general illumination inside non-residential buildings and for calculating or measuring the amount of energy required or used for lighting inside buildings.	Reliability, Accuracy, Accountability
	Relation to research topic: Similar considerations as above in terms of having baseline realistic energy performance and lighting performance against which to hold the BIM modeling accountable.		
ISO 15469:2004[12] **Spatial distribution of daylight — CIE standard general sky**	CIE International Commission on Illumination[13]	ISO 15469:2004 defines a set of outdoor daylight conditions linking sunlight and skylight for theoretical and practical purposes.	Accuracy
	Relation to research topic: Baseline reference values for general sky illumination are critical to agree on for them to be of use in multi-stakeholder BIM systems that makes predictions on usable daylight on a given time or day. This is an important component for daylight optimization.		

Table 5.3 Standards useful for real-world data collection.

Standard title	Standardization committee	Scope extract / description	Trustworthiness characteristics affected
ISO/IEC 33063:2015[14] **Information technology — Process assessment — Process assessment model for software testing**	ISO/IEC JTC 1/SC 7 Software and systems engineering[15]	ISO/IEC 33063:2015 provides guidance, by example, on the definition, selection, and use of assessment indicators. A process assessment model comprises a set of indicators of process performance and process capability. The indicators are used as a basis for collecting the objective evidence that enables an assessor to assign ratings. Essentially, the SQuaRE series offers a good general framework for quality software management, in particular requirements specification and quality evaluation.	Accuracy, Quality
	Relation to research topic: Assessment indicators offer objective evidence of software performance and quality. Such frameworks help ensure the reliability of BIM and optimization tools.		
ISO/IEC 25000:2014[16] **Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE**	ISO/IEC JTC 1/SC 7 Software and systems engineering	ISO/IEC 25000:2014 provides guidance for the use of the new series of International Standards named Systems and software Quality Requirements and Evaluation (SQuaRE). The purpose of ISO/IEC 25000:2014 is to provide a general overview of SQuaRE contents, common reference models and definitions, as well as the relationship among the documents, allowing users of the Guide a good understanding of those series of standards, according to their purpose of use.	Accuracy, Quality
	Relation to research topic: The SQuaRE series provides a transparent and structured framework for evaluating software quality. It improves the reliability of tools used in BIM and AI-supported design workflows.		

Table 5.4 Standards potentially useful for software quality assurance.

System and Output The system combines the Revit-based BIM model, parameterized window definitions, and an advanced optimization algorithm to generate near-optimal design alternatives. The building model used for experimentation is a one-storey structure of approximately 225 m^2 with eleven parameterized windows [8].

The optimization framework considers three conflicting objectives:

- **Minimize construction cost (C)**
- **Minimize energy consumption (E)**
- **Maximize daylighting (D)** in the occupied zone

Given their conflicting nature, solutions necessarily involve trade-offs.

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) was selected for its efficiency in producing diverse Pareto-optimal solutions using ranking, crowding distance, and elitism [26].

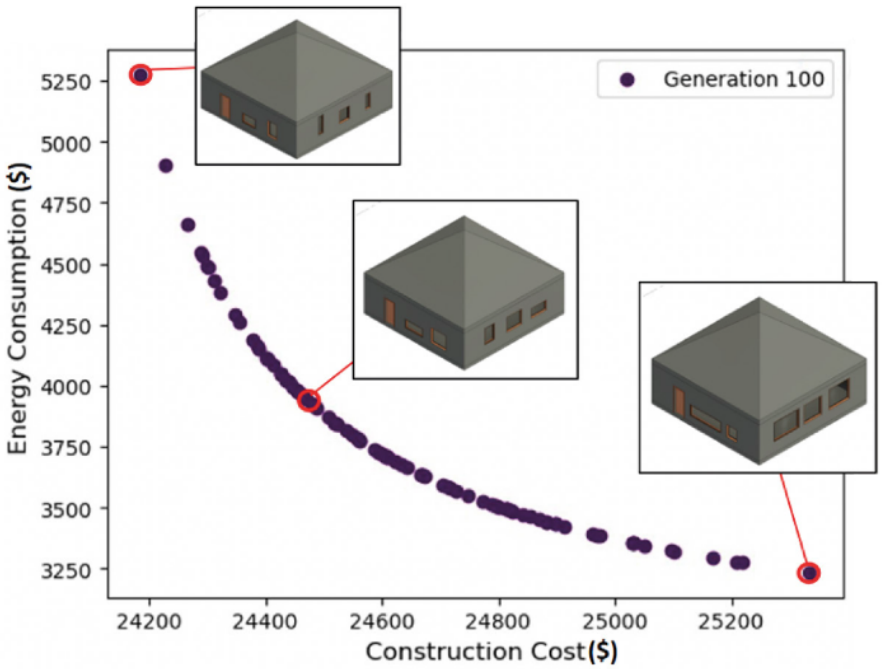


Figure 5.4 Example Pareto front for construction cost versus energy consumption, with representative building configurations illustrated at selected points along the trade-off curve.

These Pareto-optimal solutions allow decision-makers to choose the design that best aligns with their objectives and constraints.

5.2.2 Use Case Conclusion

This preliminary work demonstrates that standardization plays a crucial role in ensuring the effectiveness and trustworthiness of construction digitalization. Construction standards

help define accurate, consistent input data, and provide methodologies that enable reliable optimization across building projects.

Real-world datasets, such as energy consumption, daylight availability, or environmental conditions, further enhance model accuracy and predictive capability. Equally important is software standardization, which contributes to the quality, interoperability, and reliability of BIM-based design workflows.

These foundations motivate the transition to the next part of the chapter, which moves from BIM-driven optimization concepts to explicit CP-based modeling of a construction standard.

5.3 Integrating Construction Standards into a Constraint Programming Model

5.3.1 Motivation and Overview

This section builds on the insights developed in the previous BIM-focused use case by showing how established construction standards can be translated into formal computational models. The aim is to demonstrate how technical content traditionally expressed as textual specifications can be transformed into machine-interpretable constraints to support standards-aware design automation [63].

As a concrete example, the discussion focuses on the international standard ISO 10077-1:2017 on thermal transmittance of windows. It is used as the baseline input to define a CP model for optimizing window design with respect to energy performance and material cost [80].

The overall structure is as follows:

- introduce thermal transmittance in the context of building energy performance;
- highlight the role and relevance of ISO 10077-1:2017;
- describe how AI and CP help automate and streamline the calculations;
- present a CP model instantiation and discuss sample results.

5.4 Thermal Transmittance, Construction, and Standards

5.4.1 Thermal Transmittance and Construction

Thermal transmittance (commonly referred to as the U-value) is a key metric in building envelope design. It measures heat transfer per unit area under standardized conditions and significantly impacts energy efficiency and building comfort [75].

Each layer of a building's envelope (glazing, frame, insulation, spacers) affects overall thermal performance. Expressed in W/m^2K, the U-value depends on the thermal resistance of each layer, determined by its thickness and thermal conductivity.

Figure 5.5 illustrates the magnitude of heat loss through poorly insulated window frames, which often represent a major thermal weak point in buildings.

Figure 5.5 Thermographic image showing heat losses through window frames. Energy analyses estimate that over one third of heating energy is lost through windows and doors.

Thermographic analysis illustrates heat losses through frames and glazing connections: see Figure 5.5. Improving U-values aligns with sustainability targets in the EU Taxonomy and the EPBD directives [74], which emphasize energy efficiency and low-emission construction. Improving U-values is therefore central to meeting targets for reduced carbon footprints and enhanced building performance.

5.4.2 The Role of Technical Standards

For windows, ISO 10077-1:2017 defines the formula for computing U_w, specifying the terms, permissible ranges, and calculation procedures [80]. Manual computation is error-prone due to nonlinear interactions across materials and geometry, which motivates the use of automated CP models.

Global sustainability goals require shared and consistent methodologies for evaluating building performance. Technical standards fulfill this role by providing agreed-upon procedures and reference formulas.

For thermal transmittance of windows, ISO 10077-1:2017 specifies methods for calculating U_w based on glazing, frame, and edge properties. It defines:

- the base formula for overall window transmittance,
- definitions of all involved physical quantities,
- appropriate ranges and typical values,
- rules for selecting parameters depending on materials and configuration.

The calculations are non-trivial due to multiple interacting components. Manual computation is error-prone and time-consuming—hence the interest in AI and CP automation.

5.5 Leveraging Artificial Intelligence

AI provides tools for analyzing large datasets, identifying patterns, and supporting predictive modeling [63]. In the context of thermal analysis, AI can:

- process complex datasets from materials, climate, and usage patterns,
- detect correlations that are difficult to identify manually,
- update performance estimates dynamically as designs evolve.

Although AI alone does not replace formal building physics standards, it complements them by improving data processing and enabling real-time assessments [8].

The complexity of the thermal transmittance equation motivates modeling it as a COP, which allows the systematic exploration of design configurations under explicit constraints.

5.6 Encoding Complex Equations with Constraint Programming

CP provides a fully declarative approach for modeling ISO equations. A COP supports constraints, domain restrictions, and objective functions simultaneously [82, 69]. This ensures deterministic, reproducible evaluations aligned with ISO 10077-1.

5.7 Hyperparameter Optimization Tackling COPs

HPO explores solver-configuration spaces, such as branching heuristics, restart strategies, or propagation levels [73, 97]. Although this work does not tune the CP solver, it connects to the solver-optimization framework developed in previous chapters.

5.8 Model Instantiation and Results

5.8.1 The Thermal Transmittance Equation and Its Variables

ISO 10077-1:2017 defines the overall thermal transmittance of a window as:

$$U_w = \frac{\sum A_g U_g + \sum A_f U_f + \sum l_g \Psi_g + \sum l_{gb} \Psi_{gb}}{A_f + A_g}. \tag{5.1}$$

This weighted formulation ensures that glazing, frame, and edge effects are all represented.

The COP model encodes all relevant variables with realistic ranges, shown in Table 5.5. The objective is to minimize both thermal transmittance and cost.

5.8.2 Model Implementation

The model variables and their ranges are summarized in Table 5.5. Encoding uses XCSP3, ensuring solver portability. Constraints include:

- the ISO 10077-1 equation, - material-dependent transmittance and cost assignments, - geometric realism (frame area between 10% and 33.3% of glazing area).

The implementation uses `intension` constraints for material logic and scaling for solver stability.

Variable	Metric	Explanation	Range
A_g	m^2	Glazed area	1–5
A_f	m^2	Frame area	0.5–2
U_g	W/m^2K	Glazing transmittance	1–3
U_f	W/m^2K	Frame transmittance	2–4
Ψ_g	W/mK	Linear edge transmittance	0.03–0.1
l_g	m	Length of glazing edge	5–10
Ψ_{gb}	W/mK	Linear bar transmittance	0–0.01
l_{gb}	m	Length of glazing bar	0–1
U_w	W/m^2K	Window transmittance	0–3
M_f	–	Frame material (enum)	0–3
M_g	–	Glazing type (enum)	0–3
C_f	€	Frame cost	10–100
C_g	€	Glazing cost	20–100

Table 5.5 Variables in the COP model and their ranges.

5.8.3 Results

The Choco solver (v4.10.14) [84] finds an optimal solution with scaled objective value 111, corresponding to:

$$U_w = 1.11 \text{ W/m}^2\text{K}.$$

The optimal configuration is given in Table 5.6.

Variable	Value	Explanation
A_g	4.92	Glazed area (m^2)
A_f	0.5	Frame area (m^2)
U_g	1	Glazing transmittance (W/m^2K)
U_f	2	Frame transmittance (W/m^2K)
Ψ_g	0.03	Linear edge transmittance (W/mK)
l_g	5	Length of glazing edge (m)
Ψ_{gb}	0	Linear bar transmittance (W/mK)
l_{gb}	0	Bar length (m)
U_w	1.11	Thermal transmittance (W/m^2K)
M_f	2	PVC frame
M_g	3	Low-E glazing
C_f	40	Frame cost (€)
C_g	90	Glazing cost (€)

Table 5.6 Optimal solution returned by the COP model.

Ranges incompatible with the frame–glazing proportionality constraint correctly lead to unsatisfiable instances, demonstrating model robustness.

5.9 Conclusion and Further Prospects

This preliminary CP model shows that ISO 10077-1:2017 can be translated into a formal constraint model and solved automatically. The results demonstrate the potential for optimizing material choices and geometrical parameters to achieve better energy performance and cost efficiency.

Future extensions include:

- adding more materials and catalogue data,
- validating the model with real measurement datasets,
- extending the approach to other standards (structural, acoustic, lighting),
- integrating the COP directly with BIM workflows.

This work provides the technical foundation for the full BIM-to-CP pipeline presented in the next chapter.

5.10 Summary and Link to the Real-World Application

This chapter reviewed the interactions between BIM, AI, and optimization, and presented a preliminary CP model encoding ISO 10077-1:2017. Together, these illustrate how construction standards can be treated as computable inputs to optimization models.

While the current work focuses on a single building element, it motivates the more comprehensive approach in Chapter 6, where IFC-based BIM models and multiple engineering standards are integrated within a multi-objective CP framework for real-world design automation.

Published Work The contributions described in this chapter are published as:

- Trustworthiness in ICT, Aerospace and Construction Applications

  ILNAS-SnT collaboration (Working group: Mr Jean-Marie Reiff, Dr Jean-Philippe Humbert, Prof Dr Pascal Bouvry, Dr Grégoire Danoy, Dr Mohammed Alswaitti, Mr Manuel Combarro Simón, Ms Maria Hartmann, Ms Hedieh Haddad, Dr Lucas Cicero, Dr Jean Lancrenon, Mr Nicolas Domenjoud, Mrs Leslie Fouqueray, Mrs Natalia Vinogradova, Mr Ruddy Enguehard)

  URL: white paper « Trustworthiness in ICT, Aerospace, and Construction applications - Scientific Research and Technical Standardization »

- Research-driven Standardization Opportunities for ICT, Construction and Aerospace

  ILNAS-SnT collaboration (Working group: Dr Jean-Philippe Humbert, Prof Dr Pascal Bouvry, Dr Grégoire Danoy, Dr Mohammed Alswaitti, Mr Manuel Combarro Simón, Ms Maria Hartmann, Ms Hedieh Haddad, Dr Lucas Cicero, Dr Jean Lancrenon, Mrs Natalia Vinogradova, Mrs Victoria Mletzak)

  URL: Technical reports - Research driven approach to standardization - October 2024

Chapter 6

Real-World Applications of CP in the Construction Domain

Contents

6.1 Introduction

The AEC industry is undergoing a rapid digital transformation, driven by the growing integration of BIM and AI technologies [7, 32, 17]. Despite these advancements, a persistent challenge remains the reliable and automated interpretation of engineering standards [34, 4]. Normative documents such as ISO 10077-1 (thermal transmittance) [60], ISO 717-1 (acoustic comfort) [98], and ILNAS 103-1:2022 (building performance) [85] are essential for ensuring safety, sustainability, and regulatory compliance in building design [10]. However, these standards are typically textual and human-readable, requiring manual translation into design parameters or simulation scripts, a process that is slow, error-prone, and reactive, preventing full automation and traceable reasoning within digital workflows [4, 34].

Existing compliance-checking solutions often rely on simulation-based or rule-based systems that treat standards as black-box evaluators [24, 105, 91]. While such approaches can detect violations, they cannot *guarantee* compliance by design. Furthermore, they lack explainability and reasoning capabilities, limiting their utility in intelligent, standards-aware design automation [79, 106]. Consequently, there is a growing need for methods that enable standards to be represented, interpreted, and optimized automatically and transparently [29].

CP is a powerful paradigm for solving combinatorial problems with complex feasibility rules [89, 2]. When extended to multi-objective optimization (MOO), it offers a white-box alternative to black-box simulation-based approaches, enabling guaranteed feasibility and direct reasoning over the problem structure [44]. However, the performance of multi-objective constraint programming (MO-CP) solvers depends critically on the search strategy: variable/-value ordering heuristics influence convergence speed [45], while the ability to explore diverse regions of the objective space determines Pareto-front coverage.

This chapter investigates how to systematically enhance a base MO-CP solver through a hierarchy of search strategies. We ground our study in a challenging real-world domain: the automated, standards-compliant design of building components. This domain provides a perfect testbed: it features discrete choices, conflicting objectives (thermal performance, cost, acoustic comfort), and numerous algebraic constraints derived directly from international and national regulations (ISO, ILNAS) [60, 59, 85]. By encoding these standards as explicit constraints, we create a white-box CP model where every solution is feasible by construction [34, 90].

Instead of verifying compliance after a design is generated, we encode the standards themselves as declarative CP constraints, ensuring that every solution is inherently compliant by construction. The framework integrates BIM data from IFC files [59, 56] with formalized CP models and employs a multi-objective solver [84] to explore design trade-offs such as thermal

efficiency, acoustic performance, and cost.

Within this fixed model, we evaluate a three-level search hierarchy:

1. **Level 1: Default monolithic MO-CP** – the baseline using the solver's built-in heuristics.

2. **Level 2: Learned intensification via PSA** – a PSA [46], which uses BO to configure the solver's branching heuristics per instance, treating solver configuration as a HPO problem.

3. **Level 3: Structural exploration via LNS** – a large neighborhood search (LNS) that iteratively destroys and repairs solutions to diversify the Pareto front, using the PSA-tuned solver as its repair engine.

Our experimental testbed comprises 30 instances derived from BIM, automatically converted to CP via a dedicated pipeline. This allows us to answer key methodological questions: Does learning solver parameters (PSA) improve convergence over the default? Does LNS improve diversity? How do these strategies complement each other?

Research Question. This chapter addresses the following central research questions:

- **RQ6:** Can an integrated HPO framework be designed to generalize across different solvers, problem types, and domains?

- **RQ7:** How can engineering standards be transformed into computable constraint models that enable automated, verifiable, and standards-compliant design optimization within BIM environments?

Contributions and Scope. The main contributions of this chapter are as follows:

1. A hierarchical MO-CP search framework combining learning-based configuration (PSA) and neighborhood-based diversification (LNS), layered atop a base CP solver.

2. An empirical study on 30 real-world BIM-derived instances showing that PSA improves hypervolume by 10% over default search (intensification), while LNS significantly enhances Pareto-front coverage (diversification).

3. A detailed analysis of complementary roles: PSA excels at finding extreme solutions, whereas LNS populates interior trade-offs, together providing a more complete Pareto front.

4. A fully automated BIM-to-CP pipeline that parses IFC models, encodes ISO/ILNAS standards as CP constraints, and generates solvable MiniZinc [76] models.

5. An open-source artifact (code, models, and data) to support reproducibility and further research in computable standards and hybrid CP search.

Chapter Structure. The remainder of this chapter is organized as follows. Section 6.2 reviews related work in MO-CP, learning for optimization, LNS, and building design optimization. Section 6.3 explains the constraint-based performance model for window components. Section 6.4 details the BIM-to-CP pipeline architecture. Section 6.5 describes the three-level search hierarchy in detail. Section 6.6 outlines the experimental setup. Section 6.7 presents the results and analyses. Section 6.8 discusses the findings, interpretations for practitioners, and limitations. Finally, Section 6.9 summarizes the chapter and provides concluding remarks.

6.2 Related Work

Our work sits at the intersection of MO-CP, learning for optimization (L4O), and hybrid search strategies, applied to the domain of standards-aware building design.

6.2.1 Multi-Objective Constraint Programming

CP has been extended to multi-objective settings through dominance filtering, lexicographic ordering, and dedicated search procedures [89]. Modern solvers like Choco support MOO natively. However, few studies investigate how search heuristics and diversification strategies affect Pareto-front quality in structured design problems, particularly those derived from engineering standards.

6.2.2 Learning for Optimization and Algorithm Configuration

Automating algorithm configuration is a core topic in learning for optimization [51]. Bayesian optimization is widely used for hyperparameter tuning [37]. Our PSA approach [46] falls into this category, applying BO to configure CP solver heuristics under a fixed time budget. While typically used for single-objective problems, we adapt PSA for MO-CP by using hypervolume as the optimization target.

6.2.3 Large Neighborhood Search

LNS is a metaheuristic that iteratively destroys and repairs solutions [83]. It has been successfully hybridized with CP for single-objective optimization [82], but its application to *multi-objective* CP for design diversification is less explored. Our LNS uses domain-aware destruction operators and a PSA-tuned CP solver for repair, creating a hybrid that combines global exploration with constraint-aware local search.

6.2.4 Building Design Optimization

Most building optimization uses simulation-coupled metaheuristics (e.g., NSGA-II [26]). Constraint-based approaches are less common in this setting. While these approaches have advanced performance optimization, they remain disconnected from the logic of engineering standards. The optimizer cannot reason about regulatory thresholds or conditional relationships, since these are external to the simulation process.

Recent initiatives advocate for computable standards and rule-based reasoning integrated into digital twins [17, 33]. CP, with its declarative modeling capacity, is particularly well-suited for formalizing these standards, as its constraint logic mirrors the structure of the rules, formulas, and thresholds defined in normative documents. Our work differs by using a fully white-box CP model derived directly from standards, enabling the direct application and evaluation of advanced CP search strategies.

6.3 Constraint-Based Performance Model

We focus on a window component whose performance is evaluated through thermal transmittance, acoustic insulation, and cost. The decision variables describe geometric and material choices. We consider a multi-objective constraint satisfaction/optimization problem defined over variables x with domain X, constraints C, and objective functions $f_1(x), \ldots, f_k(x)$ to minimize. The goal is to find the set of Pareto-optimal solutions.

6.3.1 Decision Variables and Domains

For each window, we consider discrete choices of glazing type, frame type, and spacer type from a material catalog, alongside derived areas and perimeters for glazing and frame based on fixed opening geometry. The decision vector x encodes all material selections; all performance quantities are derived from x and instance-specific geometry.

6.3.2 Thermal Transmittance Constraints

The overall thermal transmittance of the window, $U_w(x)$, is computed using a weighted combination of glazing and frame conductances and linear thermal bridges (ISO 10077-1 [60]):

$$U_w(x) = \frac{A_g U_g(x) + A_f U_f(x) + l_g\, \psi_g(x)}{A_g + A_f}, \tag{6.1}$$

where A_g and A_f are glazing and frame areas, $U_g(x)$ and $U_f(x)$ the corresponding thermal transmittances, l_g the glazing perimeter, and $\psi_g(x)$ a linear thermal bridge coefficient. Thresholds on $U_w(x)$ can be encoded as linear or rational constraints over these terms.

6.3.3 Acoustic Insulation Constraints

Acoustic insulation follows a weakest-link principle (ISO 717-1 [98]):

$$R_w(x) = \min\Big(R_{w,g}(x), R_{w,f}(x)\Big), \tag{6.2}$$

where $R_{w,g}(x)$ and $R_{w,f}(x)$ denote the sound reduction indices for glazing and frame. Additional facade-level checks (e.g. based on incident noise spectra and correction terms) are encoded as linear inequalities to satisfy the regulations (ILNAS 103-1 [85]):

$$R'_w(x) + C_{\mathrm{tr}}(x) \;\geq\; R'_{w,\mathrm{req}} + \Delta, \tag{6.3}$$

where $R'_w(x)$ is a facade index, $C_{\mathrm{tr}}(x)$ a spectrum adaptation term, $R'_{w,\mathrm{req}}$ the required rating, and Δ a safety margin.

6.3.4 Cost Model

The cost objective $C(x)$ aggregates material and assembly costs associated with the selected window configuration:

$$C(x) = C_g(x) + C_f(x) + C_{\text{spacer}}(x) + C_{\text{assembly}}(x), \tag{6.4}$$

where $C_g(x)$ and $C_f(x)$ denote glazing and frame costs computed from unit prices and areas, $C_{\text{spacer}}(x)$ represents the cost of spacer elements along the glazing perimeter, and $C_{\text{assembly}}(x)$ captures fixed manufacturing and installation costs. Unit prices are taken from a synthetic but realistic catalog; the structure of (6.4) is compatible with richer, project-specific pricing models.

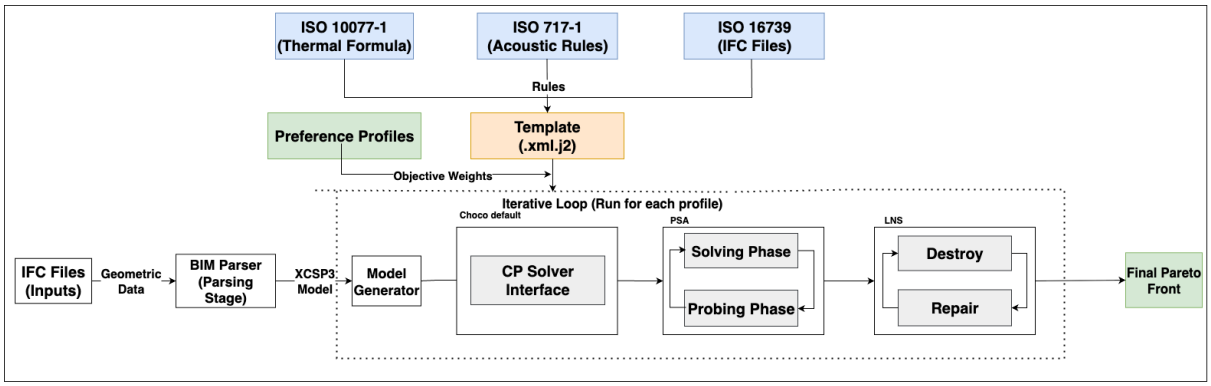


Figure 6.1 BIM-to-CP pipeline architecture. Input IFC files and standards are processed to generate a MiniZinc CP model. The solver orchestrator runs three search strategies: default CP, PSA-tuned search, and LNS. Outputs are Pareto sets and quality indicators.

6.3.5 Multi-objective Problem Statement

The multi-objective optimization problem is

$$\begin{aligned} \text{minimize} \quad & \big(U_w(x),\, C(x),\, -R_w(x)\big) \\ \text{subject to} \quad & x \in X, \\ & \text{constraints (6.1),(6.2),(6.3),(6.4) hold,} \end{aligned} \tag{6.5}$$

where X is the discrete space of admissible material choices. All performance relationships are encoded as constraints in a MiniZinc model [76], compiled to FlatZinc and solved by the Choco backend.

6.4 BIM-to-CP Pipeline

The constraint model is instantiated from BIM data via an automated pipeline implemented in Python. Figure 6.1 provides an overview of the end-to-end architecture, from standards-aware data extraction to multi-objective solving and post-processing.

The architecture is organized into:

- **Inputs:** IFC building models, material catalogs (glazing, frame, spacer), and performance standards (thermal, acoustic, facade-level).

- **BIM and standards processing:** An IFC parser based on `ifcopenshell` [55, 56] extracts window geometry and maps IFC attributes to model parameters; standards are encoded as symbolic relationships defining $U_w(x)$, $R_w(x)$, facade constraints, and $C(x)$.

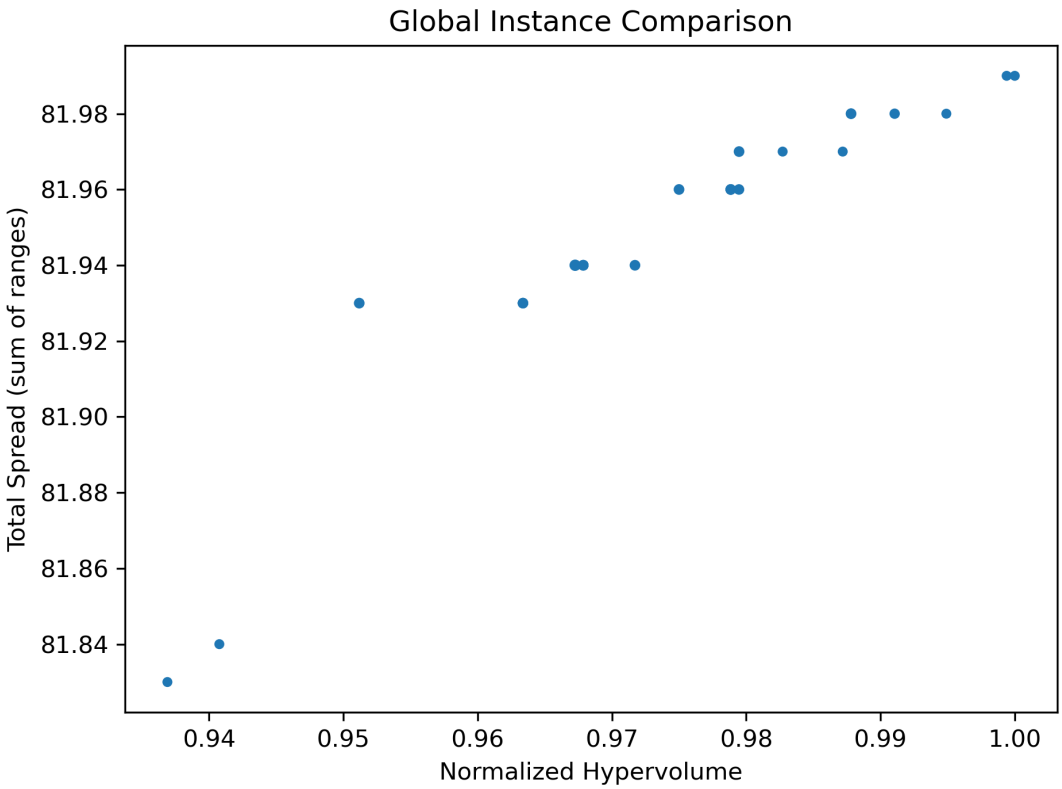


Figure 6.2 Global Pareto-spread comparison across instances. For each method, we aggregate the nondominated solutions obtained over all instances into a single set, then compute the (normalized) spread metric on this global set. Higher values indicate a broader coverage of the objective space.

- **Constraint model generation:** A parameterized MiniZinc template is instantiated using `Jinja2` and compiled to FlatZinc for Choco.

- **Solver orchestrator:** A Python controller manages the global time budget and runs (i) the default multi-objective CP search, (ii) a PSA-tuned monolithic configuration using BO, and (iii) an LNS scheme that starts from a monolithic solution and performs destroy–repair cycles.

- **Post-processing and analysis:** For each configuration and instance, nondominated solutions are merged and filtered, and quality indicators (HV and a dispersion metric) are computed [43]. In addition to HV, we report a *total spread*: for each instance, we compute the range of each objective (max–min over the nondominated set) and sum these ranges; values shown in Fig. 6.2 aggregate this quantity over all instances.

6.5 Search Strategies: A Three-Level Hierarchy

We evaluate three algorithmic configurations within the same constraint model, representing a hierarchical approach to enhancing MO-CP performance. The following strategies are evaluated within the same CP model, differing only in how they control the solver's search.

6.5.1 Level 1: Default Monolithic CP Search

The baseline uses the solver's built-in multi-objective search with default variable- and value-selection heuristics (e.g., `dom/wdeg` [18, 50]). This configuration represents the performance floor provided by the constraint model alone.

6.5.2 Level 2: Learned Intensification via PSA

To push the monolithic solver closer to its potential, we add a learning for optimization layer using the PSA. PSA treats the solver's configuration, specifically the variable selection strategy (S_{var}) and value selection strategy (S_{val}), as hyperparameters to be optimized.

The configuration space $\mathcal{P}$ combines a small but expressive set of solver parameters. In our experiments, we consider:

- **Variable selection strategy** S_{var} chosen from:
  `RunRobin`, `Wdeg`, `Memory`, `PickOnDom`, `FrOnDom`, `WdegOnDom`, `ProcOnDom`, `Regret`, `FrbaOnDom`, `Ddeg`;
- **Value selection strategy** S_{val} chosen from:
  `Dist`, `OccsR`, `Median`, `AsgsFp`, `Bivs2`, `First`, `AsgsFm`, `Last`, `Robin`, `RunRobin`, `Bivs`, `InternDist`, `Occs`, `FlrsE`;

This yields $|\mathcal{P}| = |S_{var}| \cdot |S_{val}| = 10 \times 14 = 140$ distinct configurations. PSA treats these categorical choices as hyperparameters and uses BO to maximize the expected HV of the approximated Pareto set. In our setup, PSA evaluates configurations per instance within the probing budget, selecting the best-performing configuration for the final solving phase.

In this work, we fix the restart policy to a geometric scheme, so that $|\mathcal{P}| = 10 \times 14 = 140$ distinct configurations.

Under a global time budget $T_{\max}$, PSA operates in two phases:

1. **Probing phase** ($T_{\text{probe}} \approx 0.2\, T_{\max}$): configurations $c_i \in \mathcal{P}$ are sampled, the solver is run for a short cutoff time τ_i, and the resulting HV is recorded; a Gaussian-process surrogate is updated after each evaluation.

2. **Solving phase** ($T_{\text{solve}} = T_{\max} - T_{\text{used}}$): once the probing budget is exhausted, PSA selects the configuration $c^\star$ with the highest expected improvement and runs it for the remaining time.

6.5.3 Level 3: Structural Exploration via LNS

At the third level, we introduce an LNS strategy to increase structural diversification. For each instance, we initialize LNS from the best solution found by the PSA-tuned configuration and iterate destroy–repair cycles until the global time budget $T_{\max}$ is exhausted. In each cycle, we relax 50% of the material decision variables (glazing, frame, spacer choices), fix the remaining variables, and re-optimize the resulting subproblem under a repair time limit τ using the same multi-objective CP engine. We evaluate two repair budgets, $\tau = 20$ s and $\tau = 40$ s, to study the impact of repair depth on exploration and Pareto-front coverage.

6.6 Experimental Setup

6.6.1 System Configuration and Instances

The full suite of experiments was executed on high-performance computing (HPC) facilities with the technical specifications of a cluster compute node: 2xAMD Epyc ROME 7H12 @ 2.6 GHz [64c/280W] processor with 256 GB RAM. The software environment included Python 3.9, the `ifcopenshell` and `Jinja2` libraries, and the Choco 5.0.0-beta.1 solver accessed through a custom FlatZinc wrapper. All runs were performed sequentially with a fixed random seed for each instance and configuration.

The 30 IFC instances are drawn from publicly available repositories, including the buildingSMART *Sample Test Files* collection [55], and cover a range of residential and non-residential projects. Each instance defines at least one window element with measurable geometric and material data, providing diverse opening geometries and material combinations.

6.6.2 Time Budgets and Quality Indicators

Each instance is solved under four configurations: (i) default Choco multi-objective search, (ii) PSA-tuned monolithic configuration with probing ratio 0.2, and (iii) two LNS variants with repair time limits of 20 s and 40 s. Each configuration is given a wall-clock time budget of $T_{\max} = 1200$ s per instance.

Method	Mean normalized HV
Default monolithic	0.78
PSA-tuned monolithic	0.86
LNS (40 s repair)	0.84

Table 6.1 Average normalized HV for the main configurations.

With 30 instances and four configurations per instance, the full experimental campaign comprises $30 \times 4 = 120$ runs, for a total budget of $120 \times 1200 = 144{,}000$ seconds (about 40 CPU-hours). For each instance, we compute HV on a normalized objective space to make objectives comparable. We take the union of nondominated solutions produced by all configurations on that instance, normalize each objective linearly to $[0, 1]$ using the observed min/max values, and compute HV with reference point $(1, 1, 1)$. We then normalize HV by the best value observed across all configurations so that reported scores lie in $[0, 1]$.

For each configuration and instance, we collect the nondominated solutions and compute:

- **HV** [107], normalized per instance by the best observed value;
- **Total spread**, the sum of objective-wise ranges (max–min) computed on the same normalized objectives as HV, aggregated over instances as described in Sect. 4.

6.7 Results

6.7.1 Default vs. PSA: Value of Learning

Across the 30 benchmark instances, the PSA-tuned configuration consistently improves normalized HV over the default monolithic search. Table 6.1 summarizes the average values.

The probing phase consumes approximately 20% of the global time budget ($T_{\text{probe}} \approx 0.2\, T_{\max}$), including solver runs and surrogate-model updates. Despite this overhead, PSA yields a clear increase in mean normalized HV, indicating that spending part of the budget on learning a good configuration is more beneficial than using the full budget with a suboptimal fixed heuristic.

6.7.2 PSA vs. LNS: Exploitation and Exploration

As shown in Figure 6.3, the monolithic PSA achieved the highest average HV. This is indicative of its strength as a consistent exploiter that reliably finds good, but not necessarily optimal, solutions. The performance of the LNS framework was sensitive to its repair time. A short

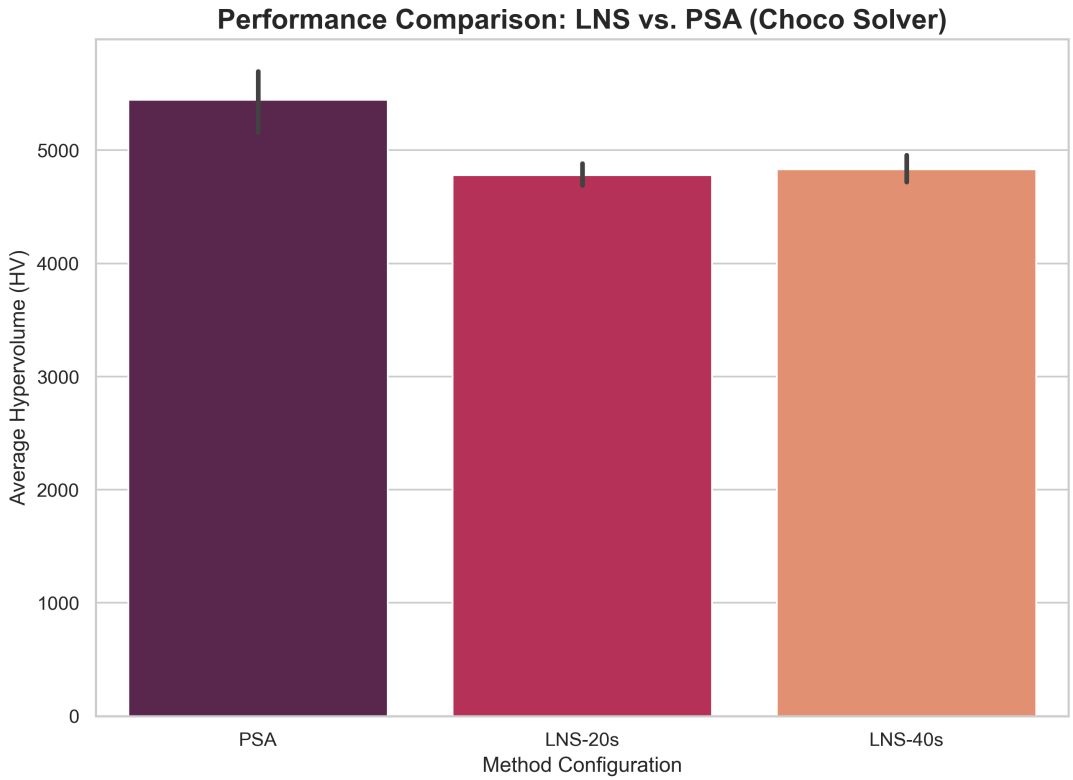


Figure 6.3 Mean HV for PSA-tuned monolithic search vs. LNS with different repair time limits (20 s, 40 s). PSA gives the highest average HV, but LNS-40 s is competitive.

20 s repair time was insufficient, leading to slightly less performance than 40 s. However, increasing the repair time to 40 s improved the LNS's average performance slightly, making it more competitive with the baseline and showing its potential as an intelligent explorer.

PSA achieves strong HV by concentrating on the extremes of the Pareto front, such as minimum cost or minimum U_w. LNS, in contrast, sacrifices some extremal performance to populate intermediate trade-offs, especially where objectives are strongly conflicting.

6.7.3 Instance-Level Case Study

To illustrate the practical impact of these search behaviors, we focus on a representative instance, *WellnesscenterSama*. Figure 6.4 compares the nondominated solutions in the (U_w, Cost) plane.

The LNS-enhanced search uncovers window configurations with intermediate U_w and cost values that achieve a better balance between energy performance and affordability.

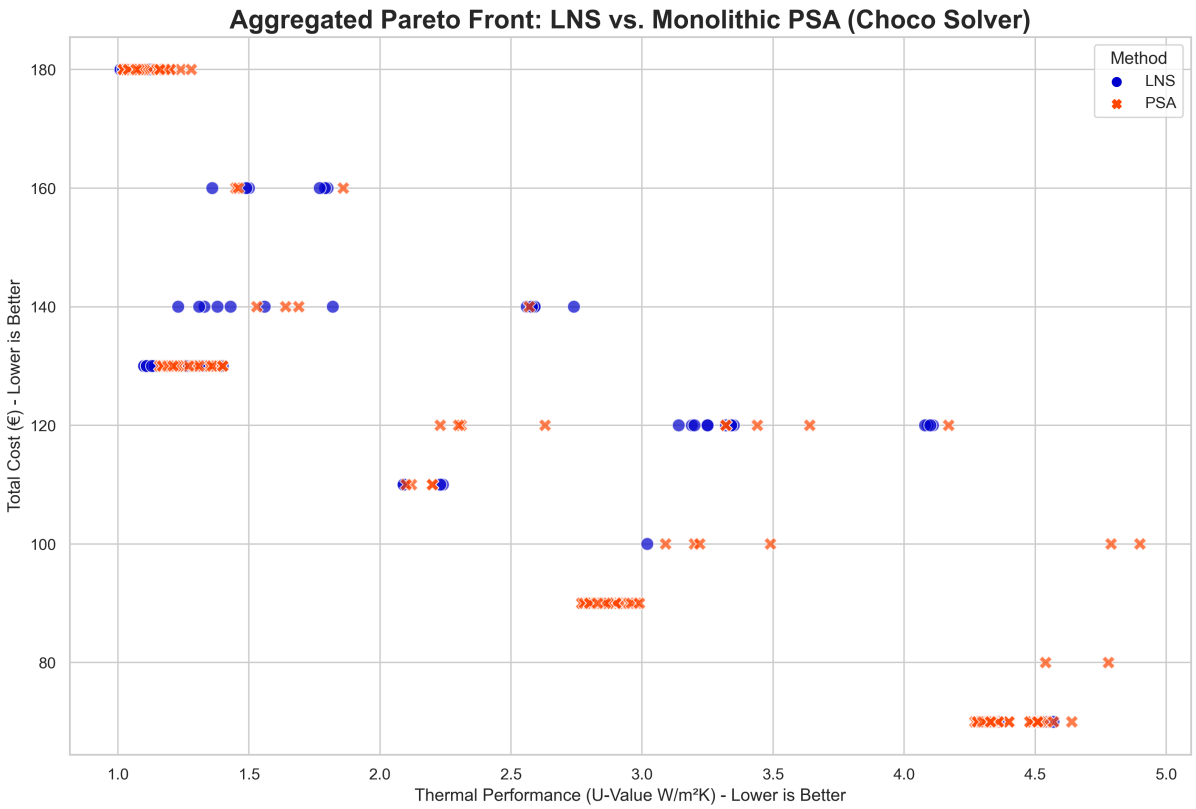


Figure 6.4 Case study Pareto front for *WellnesscenterSama* (Uw vs. Cost). PSA solutions (crosses) include the extremes (lowest cost and lowest Uw), whereas LNS solutions (circles) populate intermediate trade-offs, yielding a denser approximation of the Pareto front.

6.8 Discussion and Limitations

The experiments demonstrate a clear division of labor within our hierarchical framework. The constraint-based model ensures feasibility-by-construction, providing a rigorous foundation. At the search level, we observe a complementary relationship: the PSA-tuned monolithic search acts as an effective *exploiter*, leveraging learned heuristics to improve HV (convergence) by approximately 10% over the default solver. In contrast, LNS operates as a superior *explorer*, sacrificing a small amount of average HV to achieve greater Pareto-front diversity and coverage. This synergy, learned intensification paired with structural diversification, offers a blueprint for enhancing MO-CP solvers on structured design problems.

Interpretation for Optimization Practitioners. Our findings suggest that for highly constrained, discrete multi-objective problems, a hybrid approach is beneficial. Investing a portion of the time budget in learning solver parameters (via PSA) is worthwhile to boost baseline performance. Subsequently, employing an LNS strategy is crucial if a well-distributed set of trade-off solutions is desired, rather than just extreme points. The 20% probing overhead of PSA is justified by its consistent gains, while LNS repair budgets (e.g., 40 s) are sufficient to

allow meaningful re-optimization.

Limitations and Future Work. The study has several limitations that point to future research directions. First, the model is component-level; extending it to whole-building optimization with subsystem interactions is a natural but challenging next step. Second, the cost model is illustrative; integrating real market data and dynamic pricing would enhance practical relevance. Third, our evaluation is internal to CP strategies; a direct comparison with state-of-the-art metaheuristics like NSGA-II applied to the same constraint model would further elucidate the value of constraint propagation versus generate-and-test search. However, our primary contribution is the evaluation of CP search strategies themselves; the constraint model serves as a fixed, real-world-derived testbed. Future work will include such external baselines and investigate adaptive LNS operators and online learning mechanisms for the PSA-LNS hybrid.

6.9 Conclusion

We presented a constraint-based framework for energy-efficient component design that encodes thermal, acoustic, facade-level, and cost relationships as explicit constraints and solves the resulting multi-objective problem using a hierarchy of search strategies. The results demonstrate that constraint-based modeling provides a robust foundation for standards-compliant optimization; learning solver parameters via PSA improves intensification; and structural LNS is essential for discovering diverse, balanced trade-offs.

Future work will extend the framework to more complex building subsystems, investigate adaptive LNS operators that adjust neighborhoods and repair budgets online, and explore hybrid schemes that combine constraint-based models with reduced-order or surrogate simulations.

Part IV

Conclusion

Chapter 7

Conclusion and Future Work

Contents

7.1 Discussion

This section synthesizes the core contributions and empirical findings of this thesis, directly addressing the research questions outlined in Section 1.3.2. Through the development and evaluation of PSA—a resource-aware, solver-agnostic hyperparameter optimization framework—we demonstrate that automated configuration is both feasible and impactful for constraint programming. Our evaluation spans both solver-centric benchmarks (ACE, Choco) and applied domains (AEC via computable standards with BIM), establishing PSA's versatility.

7.1.1 Key Findings and Implications

RQ1: Applicability of HPO to CP Configuration Spaces Our results affirm that HPO methods can effectively navigate the discrete, combinatorial spaces of CP search strategies. PSA achieves this through three adaptive mechanisms: (1) time partitioning (probing/solving phases), (2) adaptive timeout evolution to gather informative signals under tight budgets, and (3) model-based navigation of categorical hyperparameters. As shown in Table 4.1, PSA consistently outperformed uninformed baselines across both ACE and Choco solvers, with solve rates reaching 93.59% and solution quality averaging 86.38% of optimal. This demonstrates that treating CP configuration as HPO is not only practical but yields measurable performance gains.

RQ2: Model-Based vs. Model-Free HPO Under Time Constraints BO emerged as the most effective approach under realistic computational limits. As detailed in Section 4.2.6, BO's surrogate modeling and acquisition-driven exploration provided superior sample efficiency, discovering high-quality configurations within short probing windows. However, our analysis reveals an important nuance: the combination of BO with *resource-aware evaluation* (adaptive timeouts) was crucial. This synergy suggests that future HPO for CP should consider both intelligent configuration selection and intelligent resource allocation as complementary concerns.

RQ3: Achieving Parameterless Robustness PSA approaches parameterless operation through conservative default settings that proved robust across varying probing ratios (5%–100% of total budget). The correlation analysis in Table 4.2 shows strong agreement ($r > 0.85$) between rankings obtained with minimal probing time and those from full-budget runs. This validation is significant: it means PSA's early screening reliably identifies promising strategies without requiring extensive parameter tuning, making it accessible to non-experts while maintaining performance.

RQ4: Generalization Across Solvers and Budgets PSA demonstrated consistent generalization across both ACE/XCSP3 and Choco/MiniZinc ecosystems. Head-to-head comparisons (Section 4.2.10) show that PSA-tuned solvers matched or surpassed their baseline counterparts in 90.38% of instances for ACE and 63.11% for Choco. The two-phase design proved particularly valuable for stability: by dedicating only a portion of the budget to exploration, PSA mitigates the risk of committing entirely to a suboptimal configuration, a critical feature for practical deployment where computational resources are finite.

RQ5: Systematic Benchmarking Reveals Strategic Patterns Our structured evaluation protocol—combining fixed budgets, per-instance champion analysis, and ablation studies—revealed a fundamental dichotomy in optimal configuration strategies. As summarized in Table 4.3.3, dynamic timeout evolution proved superior for computationally intensive optimization problems requiring deep search, while static schedules excelled for feasibility-focused constraint satisfaction tasks. This insight, only visible through systematic instance-level analysis, provides actionable guidance for both automated configuration systems and human experts.

RQ6: Cross-Domain Generalization Beyond solver configuration, PSA's modular architecture enabled successful application to the AEC domain through integration with computable standards (ISO 10077-1, ISO 717-1) and BIM data. The standards-as-constraints approach, detailed in Chapter 6, demonstrates that PSA's principles extend to real-world, multi-objective optimization with guaranteed compliance. This cross-domain validation strengthens PSA's claim as a general framework for constraint-based optimization problems.

7.1.2 Limitations and Boundary Conditions

While PSA proved effective across our evaluation, several boundary conditions merit discussion:

- **Sample efficiency trade-offs:** Although BO outperformed alternatives, its sample complexity remains non-trivial for very large configuration spaces. For such spaces, hybrid approaches combining BO with portfolio methods may be necessary.
- **Instance heterogeneity:** While PSA handles diverse problem types, instances with highly irregular search landscapes (e.g., phase transitions, extreme plateaus) may require more specialized adaptation mechanisms.
- **Warm-start limitations:** Our current implementation starts from scratch for each instance. Incorporating learned priors or warm starts from similar problems could further improve early-phase efficiency.

7.1.3 Broader Implications

The success of PSA has several implications for both research and practice:

- **Democratizing high-performance CP:** By automating configuration, PSA lowers the expertise barrier for effective CP usage, making high-performance constraint solving more accessible.

- **Towards self-configuring solvers:** PSA represents a step toward solvers that adapt their strategies based on problem characteristics, reducing reliance on manual tuning.
- **Standards as active design tools:** The AEC application demonstrates how passive standards can be transformed into active constraints, enabling optimization with guaranteed compliance—a paradigm shift for regulated domains.

Collectively, these findings establish PSA as both a practical tool for CP configuration and a conceptual framework for resource-aware algorithm design. The thesis contributes not only specific techniques and implementations but also broader insights into the interplay between configuration methods, resource allocation, and problem structure in constraint programming.

7.2 Future Research Directions

While PSA demonstrates effective HPO for CP solvers, several avenues remain open to further improve its applicability, robustness, and integration with real-world workflows. These directions are grounded in the limitations observed in our study and align with emerging trends in automated algorithm configuration and constraint-based design.

7.2.1 From Parameter-Tuning to Meta-Learning

PSA currently uses fixed meta-parameters (e.g., probing ratio ρ, timeout evolution schedule). A natural extension is to learn these parameters from instance features—such as constraint/variable ratios, objective type, or domain size—enabling *instance-aware configuration.* This would move PSA from a fixed-strategy tuner to an adaptive meta-solver that chooses not only search heuristics but also its own tuning strategy per problem class. Preliminary results already suggest that optimal timeout strategies differ between CSPs and COPs; a learned policy could automate this switch based on problem characteristics.

7.2.2 Hybrid and Warm-Start HPO for CP

Bayesian Optimization proved effective but remains sample-inefficient in very large configuration spaces. Future versions of PSA could integrate:

- **Warm starts** from historical runs or solver-specific default portfolios,
- **Multi-fidelity evaluations** that use shorter runs to prune clearly poor strategies earlier,

- **Hybrid model-free/model-based search**, where simple rules handle obvious cases and BO focuses on ambiguous regions.

This is especially relevant for industrial CP applications where similar problems recur and past configurations can bootstrap new searches, reducing the configuration overhead for recurrent problem classes.

7.2.3 Scaling and Generalization Across Solvers and Model Types

Our evaluation covered ACE and Choco within the XCSP3 and MiniZinc ecosystems. To strengthen PSA's claim of solver-agnosticism, future work should include:

- **Additional CP backends** (e.g., OR-Tools, Gecode, Picat),
- **Broader model types**, including SAT/MIP hybrids and CP-SAT,
- **Standardized benchmarking suites** with published seeds, logs, and features to support reproducibility and meta-analysis.

This would allow a more rigorous study of *portability*: identifying which aspects of configuration transfer across solvers and which remain solver-specific—a crucial step toward truly general automated configuration.

7.2.4 From Component-Level to Whole-Building Optimization in AEC

The BIM–CP pipeline demonstrated for window components shows that computable standards can drive compliant design. The next challenge is scaling to whole-building or multi-zone systems, where interactions between components (thermal bridging, shared structures, daylighting) introduce coupled constraints. This requires:

- **Hierarchical constraint models** that separate local (component) and global (system) objectives,
- **Multi-objective trade-off analysis** across energy, cost, comfort, and carbon,
- **Integration with dynamic simulation** (e.g., EnergyPlus) for time-dependent performance validation.

Such scaling would transform PSA from a component optimizer to a system-level design assistant capable of navigating the complex trade-offs inherent in building design.

7.2.5 Richer Standards and Interactive Design Guidance

Current standards (ISO 10077-1, ISO 717-1) focus on thermal and acoustic performance. Future extensions could include:

- **Structural standards** (e.g., Eurocodes),
- **Daylighting metrics** (sDA, ASE),
- **Embodied carbon rules** (EN 15978, ISO 14040).

Encoding these as constraints would support truly multi-criteria optimization. Moreover, coupling the CP backend with interactive design tools (Revit, Rhino/Grasshopper) would enable real-time, standards-aware feedback during early-stage design—shifting from *post-hoc validation* to *live compliance guidance*. This integration represents a significant step toward practical adoption in AEC workflows.

7.2.6 Trust, Explainability, and Interoperability

As automated design systems are adopted, **provenance and trust** become critical. Future work could:

- **Generate auditable compliance reports** tracing each constraint to its source standard,
- **Develop interoperable packaging** for CP models and results, enabling reuse across projects and teams,
- **Incorporate uncertainty handling** for tolerances in material properties or measurement errors, moving from deterministic to *robust* constraint models.

These enhancements would address practical concerns about transparency and reliability, facilitating wider adoption in regulated industries.

7.3 Closing Perspective

This thesis has demonstrated that treating CP solver configuration as a resource-aware HPO problem yields concrete performance gains while maintaining methodological rigor. PSA provides a principled framework that balances exploration and exploitation within fixed computational budgets, offering a practical solution to the long-standing challenge of manual strategy selection.

Beyond solver tuning, the integration of computable standards with CP opens new possibilities for automated, compliant design in regulated domains like AEC. By transforming passive standards into active constraints, we enable optimization with guaranteed compliance—a significant advance over traditional simulation-based approaches.

The research directions outlined above represent logical extensions of this work, addressing scalability, adaptability, and practical integration. As CP continues to evolve toward more automated and intelligent solving, frameworks like PSA will play an increasingly important role in making high-performance constraint solving accessible to both experts and non-experts alike.

Ultimately, this work contributes to a broader vision of intelligent computational design, where algorithms not only solve problems but also adapt their strategies to the problems at hand, learn from experience, and provide transparent, trustworthy solutions that respect both technical constraints and human design intent.

7.4 Dissemination

This section summarizes the contributions achieved during the PhD. First, peer/ editorial-reviewed outputs are enumerated, then standardisation-related publications and outreach items are listed.

7.4.1 Peer/Editorial-Reviewed Contributions

Below are all reviewed contributions completed during the PhD, in chronological order:

- **Workshop paper:** H. Haddad, P. Talbot, P. Bouvry. *Comparison of Hyperparameter Optimization Methods for Selecting Search Strategy of Constraint Programming Solvers.* PTHG-24: The Seventh Workshop on Progress Towards the Holy Grail, 2024.
- **Conference paper:** H. Haddad, P. Talbot, P. Bouvry. *Selecting Search Strategy in Constraint Solvers using Bayesian Optimization.* IEEE ICTAI 2024, pp. 764–773.

7.4.2 Standardisation-Related Publications

The following white paper and technical report were produced with ILNAS; I served as a co-author responsible for the construction domain content:

- **White paper:** *Trustworthiness in ICT, Aerospace and Construction Applications.* ILNAS, 2023.

- **Technical report:** *Research-driven Standardization Opportunities for ICT, Construction and Aerospace.* ILNAS, 2024.

7.4.3 Outreach and Ecosystem Impact

Talks, posters, and community engagement related to the thesis:

- **ILNAS World Standards Day 2022:** *Building Information Modelling (BIM) and its integration with Artificial Intelligence (AI).*
- **ILNAS World Standards Day 2023:** *Presentation of construction section of the white paper "Trustworthiness in ICT, aerospace and construction applications – Scientific research and technical standardization".*
- **ILNAS World Standards Day 2024:** *Presentation of construction section of the white paper "Research-driven Standardization Opportunities for ICT, Construction and Aerospace".*
- **University of Luxembourg Seminar 2023:** *Hyperparameter Optimization of Constraint Programming Solvers.*
- **University of Luxembourg Seminar 2024:** *Adaptive Hyperparameter Optimization for Constraint Solvers using the Probe and Solve Algorithm (PSA).*
- **SnT Partnership Day 2022, 2023, 2024 (Poster Presentation):** *Technical Standardisation for Trustworthy ICT, Aerospace and Construction.*

Bibliography

[1] Tanay Agrawal. “Bayesian Optimization”. en. In: *Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient*. Ed. by Tanay Agrawal. Berkeley, CA: Apress, 2021, pp. 81–108. ISBN: 978-1-4842-6579-6. DOI: 10.1007/978-1-4842-6579-6_4. URL: https://doi.org/10.1007/978-1-4842-6579-6 (visited on 06/18/2024).

[2] Krzysztof Apt. “Constraint propagation algorithms”. In: *Principles of Constraint Programming*. Cambridge: Cambridge University Press, 2003, pp. 254–298. ISBN: 978-0-521-82583-2. DOI: 10.1017/CBO9780511615320.007. URL: https://doi.org/10.1017/CBO9780511615320.007 (visited on 07/04/2024).

[3] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge: Cambridge University Press, 2003. ISBN: 978-0-521-82583-2. DOI: 10.1017/CBO9780511615320. URL: https://www.cambridge.org/core/books/principles-of-constraint-programming/C008FB32571F66C3EE0EEEBDE1F98A7D (visited on 01/17/2025).

[4] Joaquín Arias et al. “Building Information Modeling Using Constraint Logic Programming”. en. In: *Theory and Practice of Logic Programming* 22.5 (Sept. 2022), pp. 723–738. ISSN: 1471-0684, 1475-3081. DOI: 10.1017/S1471068422000138. URL: https://www.cambridge.org/core/journals/theory-and-practice-of-logic-programming/article/building-information-modeling-using-constraint-logic-programming/557F44DA7B93A7FAA89D967A94AF8269 (visited on 11/05/2024).

[5] Gilles Audemard, Christophe Lecoutre, and Charles Prud’homme. “Guiding Backtrack Search by Tracking Variables During Constraint Propagation”. en. In: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. DOI: 10.4230/LIPIcs.CP.2023.9. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2023.9 (visited on 06/20/2024).

[6] Charles Audet et al. “Performance indicators in multiobjective optimization”. In: *European Journal of Operational Research* 292.2 (July 2021), pp. 397–422. ISSN: 0377-2217.

DOI: 10.1016/j.ejor.2020.11.016. URL: https://www.sciencedirect.com/science/article/pii/S0377221720309620 (visited on 11/05/2025).

[7] Salman Azhar and Justin Brown. "BIM for Sustainability Analyses". en. In: *International Journal of Construction Education and Research* 5.4 (Nov. 2009), pp. 276–292. ISSN: 1557-8771, 1550-3984. DOI: 10.1080/15578770903355657. URL: http://www.tandfonline.com/doi/abs/10.1080/15578770903355657 (visited on 04/14/2023).

[8] Minjung Bae et al. "Analysis of the Calculation Method for the Thermal Transmittance of Double Windows Considering the Thermal Properties of the Air Cavity". en. In: *Sustainability* 12.24 (Jan. 2020). Publisher: Multidisciplinary Digital Publishing Institute, p. 10439. ISSN: 2071-1050. DOI: 10.3390/su122410439. URL: https://www.mdpi.com/2071-1050/12/24/10439 (visited on 11/20/2025).

[9] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. en. Amsterdam: Springer Science & Business Media, July 2001. ISBN: 978-0-7923-7408-4.

[10] Thomas H. Beach, Jean-Laurent Hippolyte, and Yacine Rezgui. "Towards the adoption of automated regulatory compliance checking in the built environment". In: *Automation in Construction* 118 (Oct. 2020), p. 103285. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2020.103285. URL: https://www.sciencedirect.com/science/article/pii/S0926580519310726 (visited on 01/08/2026).

[11] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *J. Mach. Learn. Res.* 13.null (Feb. 2012), pp. 281–305. ISSN: 1532-4435.

[12] James Bergstra et al. "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems*. Vol. 24. Cambridge, US: Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper-files/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html (visited on 03/28/2025).

[13] Christian Bessiere. "Chapter 3 - Constraint Propagation". In: *Foundations of Artificial Intelligence*. Ed. by Francesca Rossi, Peter van Beek, and Toby Walsh. Vol. 2. Handbook of Constraint Programming. Amsterdam: Elsevier, Jan. 2006, pp. 29–83. DOI: 10.1016/S1574-6526(06)80007-6. URL: https://www.sciencedirect.com/science/article/pii/S1574652606800076 (visited on 10/15/2025).

[14] Christian Bessiere et al. "The complexity of global constraints". In: *Proceedings of the 19th national conference on Artifical intelligence*. AAAI'04. San Jose, California: AAAI Press, July 2004, pp. 112–117. ISBN: 978-0-262-51183-4. (Visited on 01/01/2026).

[15] Bernd Bischl et al. *Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges*. arXiv:2107.05847 [stat]. Nov. 2021. DOI: 10.48550/arXiv.2107.05847. URL: http://arxiv.org/abs/2107.05847 (visited on 05/20/2025).

[16] Ignace Bleukx et al. "Model-Based Algorithm Configuration with Adaptive Capping and Prior Distributions". en. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Ed. by Pierre Schaus. Vol. 13292. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 64–73. ISBN: 978-3-031-08010-4. DOI: 10.1007/978-3-031-08011-1. URL: https://link.springer.com/10.1007/978-3-031-08011-1 (visited on 03/28/2025).

[17] Marzia Bolpagni et al. "An Explorative Analysis of European Standards on Building Information Modelling". English. In: *Proceedings of the 2022 European Conference on Computing in Construction*. European Council on Computing in Construction (EC3), July 2022. DOI: 10.35490/EC3.2022.170. URL: https://www.research.ed.ac.uk/en/publications/an-explorative-analysis-of-european-standards-on-building-informa (visited on 09/11/2025).

[18] Frederic Boussemart et al. "Boosting systematic search by weighting constraints". en. In: (Jan. 2004). URL: https://api.semanticscholar.org/CorpusID:17864847.

[19] Frederic Boussemart et al. *XCSP3: An Integrated Format for Benchmarking Combinatorial Constrained Problems*. arXiv:1611.03398 [cs]. Nov. 2022. DOI: 10.48550/arXiv.1611.03398. URL: http://arxiv.org/abs/1611.03398 (visited on 01/17/2024).

[20] Frédéric Boussemart et al. *XCSP3-core: A Format for Representing Constraint Satisfaction/Optimization Problems*. arXiv:2009.00514 [cs]. Aug. 2024. DOI: 10.48550/arXiv.2009.00514. URL: http://arxiv.org/abs/2009.00514 (visited on 04/16/2025).

[21] Eric Brochu, Vlad M. Cora, and Nando de Freitas. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. arXiv:1012.2599 [cs]. Dec. 2010. DOI: 10.48550/arXiv.1012.2599. URL: http://arxiv.org/abs/1012.2599 (visited on 01/09/2026).

[22] Antonio Candelieri. *Mastering the exploration-exploitation trade-off in Bayesian Optimization*. arXiv:2305.08624 [cs, math]. May 2023. DOI: 10.48550/arXiv.2305.08624. URL: http://arxiv.org/abs/2305.08624 (visited on 06/27/2024).

[23] *CPMpy: Constraint Programming and Modeling in Python — CPMpy 0.9.24 documentation*. URL: https://cpmpy.readthedocs.io/en/latest/ (visited on 10/16/2025).

[24] Drury B. Crawley et al. “EnergyPlus: Creating a new-generation building energy simulation program”. In: *Energy and Buildings* 33.4 (Apr. 2001), pp. 319–331. ISSN: 0378-7788. DOI: 10.1016/S0378-7788(00)00114-6. URL: http://www.scopus.com/inward/record.url?scp=0034825352&partnerID=8YFLogxK (visited on 09/11/2025).

[25] Philipp Danzinger et al. “Solving the Test Laboratory Scheduling Problem with Variable Task Grouping”. en. In: *Proceedings of the International Conference on Automated Planning and Scheduling* 30 (June 2020), pp. 357–365. ISSN: 2334-0843. DOI: 10.1609/icaps.v30i1.6681. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/6681 (visited on 01/09/2026).

[26] K. Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. en. In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002), pp. 182–197. ISSN: 1089778X. DOI: 10.1109/4235.996017. URL: http://ieeexplore.ieee.org/document/996017/ (visited on 11/20/2025).

[27] Kalyan Deb. “Multiobjective Optimization Using Evolutionary Algorithms. Wiley, New York”. In: Jan. 2001.

[28] Rina Dechter. *Constraint Processing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Apr. 2003. ISBN: 978-0-08-050295-3.

[29] Mikhail Demianenko and Carlo Iapige De Gaetani. “A Procedure for Automating Energy Analyses in the BIM Context Exploiting Artificial Neural Networks and Transfer Learning Technique”. en. In: *Energies* 14.10 (May 2021), p. 2956. ISSN: 1996-1073. DOI: 10.3390/en14102956. URL: https://www.mdpi.com/1996-1073/14/10/2956 (visited on 05/02/2023).

[30] Janez Demsar. “Statistical Comparisons of Classifiers over Multiple Data Sets”. In: *Journal of Machine Learning Research* 7 (Jan. 2006), pp. 1–30.

[31] Yadolah Dodge. “Spearman Rank Correlation Coefficient”. en. In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer, 2008, pp. 502–505. ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_379. URL: https://doi.org/10.1007/978-0-387-32833-1_379 (visited on 01/17/2024).

[32] Yuhan Dong et al. “Intelligent optimization: A novel framework to automatize multi-objective optimization of building daylighting and energy performances”. In: *Journal of Building Engineering* 43 (2021), p. 102804. ISSN: 2352-7102. DOI: https://doi.org/10.1016/j.jobe.2021.102804. URL: https://www.sciencedirect.com/science/article/pii/S2352710221006628.

[33] Philip Dunbavin. “SUSTAINABLE ACOUSTIC DESIGN IN CONSTRUCTION -A HOLISTIC APPROACH”. In: (). URL: https://www.academia.edu/66334547/SUSTAINABLE_ACOUSTIC_DESIGN_IN_CONSTRUCTION_A_HOLISTIC_APPROACH (visited on 09/11/2025).

[34] C. Eastman et al. “Automatic rule-based checking of building designs”. In: *Automation in Construction* 18.8 (Dec. 2009), pp. 1011–1033. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2009.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0926580509001198 (visited on 09/11/2025).

[35] Katharina Eggensperger et al. “Efficient Benchmarking of Hyperparameter Optimizers via Surrogates”. en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 29.1 (Feb. 2015). Number: 1. ISSN: 2374-3468. DOI: 10.1609/aaai.v29i1.9375. URL: https://ojs.aaai.org/index.php/AAAI/article/view/9375 (visited on 11/25/2024).

[36] Stefan Falkner, Aaron Klein, and Frank Hutter. *BOHB: Robust and Efficient Hyperparameter Optimization at Scale*. arXiv:1807.01774 [cs]. July 2018. DOI: 10.48550/arXiv.1807.01774. URL: http://arxiv.org/abs/1807.01774 (visited on 11/10/2025).

[37] Matthias Feurer and Frank Hutter. “Hyperparameter Optimization”. en. In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. The Springer Series on Challenges in Machine Learning. Cham: Springer International Publishing, 2019, pp. 3–33. ISBN: 978-3-030-05318-5. DOI: 10.1007/978-3-030-05318-5_1. URL: https://doi.org/10.1007/978-3-030-05318-5 (visited on 01/13/2024).

[38] P. Frazier. “A Tutorial on Bayesian Optimization”. In: *ArXiv* (July 2018). URL: https://www.semanticscholar.org/paper/A-Tutorial-on-Bayesian-Optimization-Frazier/c27078d60737ea10e8ca4f05acd114fef29c8276 (visited on 01/08/2026).

[39] Paulo Paneque Galuzio et al. “MOBOpt — multi-objective Bayesian optimization”. In: *SoftwareX* 12 (July 2020), p. 100520. ISSN: 2352-7110. DOI: 10.1016/j.softx.2020.100520. URL: https://www.sciencedirect.com/science/article/pii/S2352711020300911 (visited on 11/11/2025).

[40] Jielong Gan et al. “Parametric BIM-Based Lifecycle Performance Prediction and Optimisation for Residential Buildings Using Alternative Materials and Designs”. en. In: *Buildings* 13.4 (Apr. 2023). Publisher: Multidisciplinary Digital Publishing Institute,

p. 904. ISSN: 2075-5309. DOI: 10.3390/buildings13040904. URL: https://www.mdpi.com/2075-5309/13/4/904 (visited on 11/20/2025).

[41] Weiao Gan, Ziyuan Ji, and Yongqing Liang. "Acquisition Functions in Bayesian Optimization". In: *2021 2nd International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*. Sept. 2021, pp. 129–135. DOI: 10.1109/ICBASE53849.2021.00032. URL: https://ieeexplore.ieee.org/document/9696089 (visited on 05/20/2025).

[42] Carla P. Gomes, Bart Selman, and Nuno Crato. "Heavy-tailed distributions in combinatorial search". en. In: *Principles and Practice of Constraint Programming-CP97*. Ed. by Gert Smolka. Berlin, Heidelberg: Springer, 1997, pp. 121–135. ISBN: 978-3-540-69642-1. DOI: 10.1007/BFb0017434.

[43] Andreia P. Guerreiro, Carlos M. Fonseca, and Luís Paquete. "The Hypervolume Indicator: Problems and Algorithms". In: *ACM Computing Surveys* 54.6 (July 2022). arXiv:2005.00515 [cs], pp. 1–42. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3453474. URL: http://arxiv.org/abs/2005.00515 (visited on 10/09/2025).

[44] Nyoman Gunantara. "A review of multi-objective optimization: Methods and its applications". In: *Cogent Engineering* 5.1 (Jan. 2018). Ed. by Qingsong Ai. Publisher: Cogent OA _eprint: https://doi.org/10.1080/23311916.2018.1502242, p. 1502242. ISSN: null. DOI: 10.1080/23311916.2018.1502242. URL: https://doi.org/10.1080/23311916.2018.1502242 (visited on 01/08/2026).

[45] Hedieh Haddad, Pierre Talbot, and Pascal Bouvry. "Comparison of Hyperparameter Optimization Methods for Selecting Search Strategy of Constraint Programming Solvers". English. In: *Progress Towards the Holy Grail*. July 2024. URL: https://orbilu.uni.lu/handle/10993/63582 (visited on 12/01/2025).

[46] Hedieh Haddad, Pierre Talbot, and Pascal Bouvry. "Selecting Search Strategy in Constraint Solvers using Bayesian Optimization". In: *2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI)*. ISSN: 2375-0197. Oct. 2024, pp. 764–773. DOI: 10.1109/ICTAI62512.2024.00113. URL: https://ieeexplore.ieee.org/abstract/document/10849432 (visited on 08/01/2025).

[47] Shai Haim and Marijn Heule. *Towards Ultra Rapid Restarts*. arXiv:1402.4413 [cs]. Feb. 2014. DOI: 10.48550/arXiv.1402.4413. URL: http://arxiv.org/abs/1402.4413 (visited on 07/04/2024).

[48] Yuanyuan Hao et al. "Constrained multi-objective optimization problems: Methodologies, algorithms and applications". In: *Knowledge-Based Systems* 299 (Sept. 2024), p. 111998. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2024.111998. URL: https://www.sciencedirect.com/science/article/pii/S0950705124006324 (visited on 11/02/2025).

[49] J. N. Hooker and W.-J. Van Hoeve. "Constraint programming and operations research". en. In: *Constraints* 23.2 (Apr. 2018), pp. 172–195. ISSN: 1383-7133, 1572-9354. DOI: 10.1007/s10601-017-9280-3. URL: http://link.springer.com/10.1007/s10601-017-9280-3 (visited on 01/17/2025).

[50] Wattez Hugues et al. "Refining Constraint Weighting". In: Nov. 2019, pp. 71–77. DOI: 10.1109/ICTAI.2019.00019. URL: https://doi.org/10.1109/ICTAI.2019.00019.

[51] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. "An Efficient Approach for Assessing Hyperparameter Importance". en. In: *Proceedings of the 31st International Conference on Machine Learning*. ISSN: 1938-7228. PMLR, Jan. 2014, pp. 754–762. URL: https://proceedings.mlr.press/v32/hutter14.html (visited on 11/25/2024).

[52] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". en. In: *Learning and Intelligent Optimization*. Ed. by Carlos A. Coello Coello. Berlin, Heidelberg: Springer, 2011, pp. 507–523. ISBN: 978-3-642-25566-3. DOI: 10.1007/978-3-642-25566-3_40.

[53] Frank Hutter, Holger H. Hoos, and Thomas Stützle. "Automatic algorithm configuration based on local search". In: *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*. AAAI'07. Vancouver, British Columbia, Canada: AAAI Press, July 2007, pp. 1152–1157. ISBN: 978-1-57735-323-2. (Visited on 11/25/2024).

[54] Frank Hutter et al. "ParamILS: An Automatic Algorithm Configuration Framework". In: *Journal of Artificial Intelligence Research* 36 (Oct. 2009). arXiv:1401.3492 [cs], pp. 267–306. ISSN: 1076-9757. DOI: 10.1613/jair.2861. URL: http://arxiv.org/abs/1401.3492 (visited on 01/21/2025).

[55] *IFC Examples*. en-GB. URL: https://technical.buildingsmart.org/standards/ifc/ifc-examples/ (visited on 12/19/2025).

[56] *IfcOpenShell - The open source IFC toolkit and geometry engine*. URL: https://ifcopenshell.org/ (visited on 09/10/2025).

[57] *Industry Foundation Classes (IFC) - buildingSMART International*. en-GB. Running Time: 4937. Nov. 2024. URL: https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/ (visited on 11/20/2025).

[58] Gabriele Iommazzo et al. "The Algorithm Configuration Problem". In: arXiv:2403.00898 [cs]. 2023, pp. 1–8. DOI: 10.1007/978-3-030-54621-2_749-1. URL: http://arxiv.org/abs/2403.00898 (visited on 11/10/2025).

[59] *ISO 10077-1:2017*. en. URL: https://www.iso.org/standard/67090.html (visited on 09/08/2025).

[60] *ISO 16739-1:2024*. en. URL: https://www.iso.org/standard/84123.html (visited on 09/08/2025).

[61] Donald R. Jones, Matthias Schonlau, and William J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". en. In: *Journal of Global Optimization* 13.4 (Dec. 1998), pp. 455–492. ISSN: 1573-2916. DOI: 10.1023/A:1008306431147. URL: https://doi.org/10.1023/A:1008306431147 (visited on 11/11/2025).

[62] Hans Kellerer, Ulrich Pferschy, and David Pisinger. "The Bounded Knapsack Problem". en. In: *Knapsack Problems*. Ed. by Hans Kellerer, Ulrich Pferschy, and David Pisinger. Berlin, Heidelberg: Springer, 2004, pp. 185–209. ISBN: 978-3-540-24777-7. DOI: 10.1007/978-3-540-24777-7_7. URL: https://doi.org/10.1007/978-3-540-24777-7_7 (visited on 01/09/2026).

[63] Ayaz Ahmad Khan et al. "Integrating Building Information Modelling and Artificial Intelligence in Construction Projects: A Review of Challenges and Mitigation Strategies". en. In: *Technologies* 12.10 (Oct. 2024). Publisher: Multidisciplinary Digital Publishing Institute, p. 185. ISSN: 2227-7080. DOI: 10.3390/technologies12100185. URL: https://www.mdpi.com/2227-7080/12/10/185 (visited on 10/10/2025).

[64] Lars Kotthoff. *Constraint solvers: An empirical evaluation of design decisions*. arXiv:1002.0134 [cs]. Jan. 2010. DOI: 10.48550/arXiv.1002.0134. URL: http://arxiv.org/abs/1002.0134 (visited on 11/20/2025).

[65] Lars Kotthoff et al. "Auto-WEKA: Automatic Model Selection and Hyperparameter Optimization in WEKA". en. In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Cham: Springer

International Publishing, 2019, pp. 81–95. ISBN: 978-3-030-05318-5. DOI: 10.1007/978-3-030-05318-5_4. URL: https://doi.org/10.1007/978-3-030-05318-5_4 (visited on 10/15/2025).

[66] H. J. Kushner. "A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise". en. In: *Journal of Basic Engineering* 86.1 (Mar. 1964), pp. 97–106. ISSN: 0021-9223. DOI: 10.1115/1.3653121. URL: https://asmedigitalcollection.asme.org/fluidsengineering/article/86/1/97/392213/A-New-Method-of-Locating-the-Maximum-Point-of-an (visited on 05/20/2025).

[67] Christophe Lecoutre. *ACE, a generic constraint solver*. en. arXiv:2302.05405 [cs]. Jan. 2023. URL: http://arxiv.org/abs/2302.05405 (visited on 06/20/2024).

[68] Christophe Lecoutre. *ACE, a generic constraint solver*. arXiv:2302.05405 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2302.05405. URL: http://arxiv.org/abs/2302.05405 (visited on 04/16/2025).

[69] Christophe Lecoutre. *Constraint networks: techniques and algorithms*. en. Hoboken, NJ: Wiley, 2009. ISBN: 978-1-84821-106-3. DOI: 10.1002/9780470611821.

[70] KU Leuven. "Things we underestimated while developing the CPMpy constraint modelling library". en. In: (2020). URL: https://modref.github.io/papers/ModRef2023_ThingsWeUnderestimatedWhileDevelopingTheCPMpyConstraintModellingLibrary.pdf.

[71] Lisha Li et al. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. arXiv:1603.06560 [cs, stat]. June 2018. DOI: https://doi.org/10.48550/arXiv.1603.06560. URL: http://arxiv.org/abs/1603.06560 (visited on 01/13/2024).

[72] Petro Liashchynskyi and Pavlo Liashchynskyi. *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS*. arXiv:1912.06059 [cs, stat]. Dec. 2019. DOI: 10.48550/arXiv.1912.06059. URL: http://arxiv.org/abs/1912.06059 (visited on 01/13/2024).

[73] Manuel López-Ibáñez et al. "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (Jan. 2016), pp. 43–58. ISSN: 2214-7160. DOI: 10.1016/j.orp.2016.09.002. URL: https://www.sciencedirect.com/science/article/pii/S2214716015300270 (visited on 11/10/2025).

[74] Duc Long Luong et al. “Developing the hybrid BIM-BEM and jellyfish search optimization system for optimizing energy consumption and building installation costs”. en. In: *Scientific Reports* 14.1 (July 2024). Publisher: Nature Publishing Group, p. 17186. ISSN: 2045-2322. DOI: 10.1038/s41598-024-68021-6. URL: https://www.nature.com/articles/s41598-024-68021-6 (visited on 11/20/2025).

[75] Maria Malvoni et al. “CFD modeling to evaluate the thermal performances of window frames in accordance with the ISO 10077”. In: *Energy* 111 (Sept. 2016), pp. 430–438. ISSN: 0360-5442. DOI: 10.1016/j.energy.2016.06.002. URL: https://www.sciencedirect.com/science/article/pii/S0360544216307745 (visited on 11/20/2025).

[76] Nicholas Nethercote et al. “MiniZinc: Towards a Standard CP Modelling Language”. en. In: *Principles and Practice of Constraint Programming – CP 2007*. Ed. by Christian Bessière. Berlin, Heidelberg: Springer, 2007, pp. 529–543. ISBN: 978-3-540-74970-7. DOI: 10.1007/978-3-540-74970-7_38. URL: https://www.doi.org/10.1007/978-3-540-74970-7_38.

[77] Eoin O’Mahony et al. “Using case-based reasoning in an algorithm portfolio for constraint solving”. In: *Irish conference on artificial intelligence and cognitive science*. Issue: 05. 2008, pp. 210–216. URL: https://www.academia.edu/download/3459808/cpHydra.pdf (visited on 01/08/2026).

[78] Anthony Palmieri and Guillaume Perez. “Objective as a Feature for Robust Search Strategies”. en. In: *Principles and Practice of Constraint Programming*. Ed. by John Hooker. Cham: Springer International Publishing, 2018, pp. 328–344. ISBN: 978-3-319-98334-9. DOI: 10.1007/978-3-319-98334-9_22. URL: https://www.doi.org/10.1007/978-3-319-98334-9_22.

[79] Yiqun Pan et al. “Building energy simulation and its application for building performance optimization: A review of methods, tools, and case studies”. In: *Advances in Applied Energy* 10 (June 2023), p. 100135. ISSN: 2666-7924. DOI: 10.1016/j.adapen.2023.100135. URL: https://www.sciencedirect.com/science/article/pii/S2666792423000148 (visited on 09/11/2025).

[80] Laura Papaleo. “Introduction to XML and its applications”. In: Jan. 2013, pp. 109–140. DOI: 10.1142/9789812836304_0006.

[81] Laurent Perron, Frédéric Didier, and Steven Gay. “The CP-SAT-LP Solver”. In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Ed. by Roland H. C. Yap. Vol. 280. Leibniz International Proceedings in Informatics

(LIPIcs). ISSN: 1868-8969. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 3:1–3:2. ISBN: 978-3-95977-300-3. DOI: 10.4230/LIPIcs.CP.2023.3. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2023.3 (visited on 01/08/2026).

[82] Gilles Pesant and Michel Gendreau. "View of local search in constraint programming". en. In: vol. 1118. Cambridge, US: Springer-Verlag, 1996, pp. 353–366. ISBN: 978-3-540-61551-4. DOI: 10.1007/3-540-61551-2. URL: https://doi.org/10.1007/3-540-61551-2 (visited on 10/15/2025).

[83] David Pisinger and Stefan Ropke. "Large Neighborhood Search". en. In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. Boston, MA: Springer US, 2010, pp. 399–419. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_13. URL: https://doi.org/10.1007/978-1-4419-1665-5_13 (visited on 09/10/2025).

[84] Charles Prud'homme and Jean-Guillaume Fages. *Choco-solver: A Java library for constraint programming*. Issue: 78 Pages: 4708 Publication Title: Journal of Open Source Software Volume: 7 original-date: 2011-11-04. 2022. DOI: 10.21105/joss.04708. URL: https://github.com/chocoteam/choco-solver (visited on 06/27/2024).

[85] *Publication de la norme nationale ILNAS 103-1:2022 relative à l'acoustique dans les bâtiments d'habitation*. fr. Mar. 2022. URL: http://portail-qualite.public.lu/fr/actualites/normes-normalisation/2022/PublicationnormeAcoustique.html (visited on 10/09/2025).

[86] Llukan Puka. "Kendall's Tau". en. In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer, 2011, pp. 713–715. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_324. URL: https://doi.org/10.1007/978-3-642-04898-2_324 (visited on 01/17/2024).

[87] Carl Edward Rasmussen. "Gaussian Processes in Machine Learning". en. In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*. Ed. by Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch. Berlin, Heidelberg: Springer, 2004, pp. 63–71. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_4. URL: https://doi.org/10.1007/978-3-540-28650-9 (visited on 05/20/2025).

[88] Jean-Charles Régin. *Generalized Arc Consistency for Global Cardinality Constraint*. Journal Abbreviation: Proceedings AAAI'96 Pages: 215 Publication Title: Proceedings AAAI'96. Jan. 1996.

[89] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming [Book]*. en. ISBN: 9780080463803. Aug. 2006. URL: https://www.oreilly.com/library/view/handbook-of-constraint/9780444527264/ (visited on 06/18/2024).

[90] Rafael Sacks et al. *BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers*. Aug. 2018. ISBN: 978-1-119-28753-7. DOI: 10.1002/9781119287568.

[91] Imran Saeed. "BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors". In: (Apr. 2013). URL: https://www.academia.edu/3183272/BIM_handbook_A_guide_to_building_information_modeling_for_owners_managers_designers_engineers_and_contractors (visited on 09/11/2025).

[92] Tom Schrijvers et al. "Search combinators". In: *Springer, Berlin, Heidelberg* 18.2 (2013), pp. 269–305. URL: http://link.springer.com/article/10.1007/s10601-012-9137-8 (visited on 03/09/2015).

[93] *scikit-optimize: sequential model-based optimization in Python — scikit-optimize 0.9.0 documentation*. URL: https://scikit-optimize.github.io/dev/ (visited on 06/26/2024).

[94] Bobak Shahriari et al. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1 (Jan. 2016), pp. 148–175. ISSN: 1558-2256. DOI: 10.1109/JPROC.2015.2494218. URL: https://ieeexplore.ieee.org/document/7352306 (visited on 01/08/2026).

[95] Helmut Simonis and Barry O'Sullivan. "Search Strategies for Rectangle Packing". en. In: *Principles and Practice of Constraint Programming*. Ed. by Peter J. Stuckey. Berlin, Heidelberg: Springer, 2008, pp. 52–66. ISBN: 978-3-540-85958-1. DOI: 10.1007/978-3-540-85958-1_4. URL: https://doi.org/10.1007/978-3-540-85958-1_4.

[96] Aleksandrs Slivkins. *Introduction to Multi-Armed Bandits*. arXiv:1904.07272 [cs, stat]. Apr. 2024. DOI: 10.48550/arXiv.1904.07272. URL: http://arxiv.org/abs/1904.07272 (visited on 06/21/2024).

[97] Niranjan Srinivas et al. *Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design*. en. Dec. 2009. DOI: 10.1109/TIT.2011.2182033. URL: https://arxiv.org/abs/0912.3995v4 (visited on 11/11/2025).

[98] International Standard. *ISO 717-1:2020*. en. URL: https://www.iso.org/standard/77435.html (visited on 09/08/2025).

[99] Peter J. Stuckey et al. “The MiniZinc Challenge 2008–2013”. en. In: *AI Magazine* 35.2 (June 2014). Number: 2, pp. 55–60. ISSN: 2371-9621. DOI: 10.1609/aimag.v35i2.2539. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2539 (visited on 01/18/2024).

[100] Erich Teppan, Gerhard Friedrich, and Andreas Falkner. “QuickPup: A Heuristic Backtracking Algorithm for the Partner Units Configuration Problem”. en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 26.2 (July 2012), pp. 2329–2334. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v26i2.18979. URL: https://doi.org/10.1609/aaai.v26i2.18979 (visited on 07/12/2023).

[101] *The MiniZinc Handbook — The MiniZinc Handbook 2.3.0*. URL: https://docs.minizinc.dev/en/2.3.0/index.html (visited on 06/20/2024).

[102] Joaquin Vanschoren. *Meta-Learning: A Survey*. arXiv:1810.03548 [cs]. Oct. 2018. DOI: 10.48550/arXiv.1810.03548. URL: http://arxiv.org/abs/1810.03548 (visited on 12/17/2025).

[103] Sebastien Varrette et al. “Management of an Academic HPC & Research Computing Facility: The ULHPC Experience 2.0”. In: *Proceedings of the 2022 6th High Performance Computing and Cluster Technologies Conference*. HPCCT ’22. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 14–24. ISBN: 978-1-4503-9664-6. URL: https://doi.org/10.1145/3560442.3560445 (visited on 06/25/2024).

[104] Andreas Walch et al. “BEMTrace: Visualization-driven approach for deriving Building Energy Models from BIM”. In: *IEEE Transactions on Visualization and Computer Graphics* 31.1 (Jan. 2025). arXiv:2407.19464 [cs], pp. 240–250. ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: 10.1109/TVCG.2024.3456315. URL: http://arxiv.org/abs/2407.19464 (visited on 11/20/2025).

[105] Gregory J. Ward. “The RADIANCE lighting simulation and rendering system”. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. SIGGRAPH ’94. New York, NY, USA: Association for Computing Machinery, July 1994, pp. 459–472. ISBN: 978-0-89791-667-7. DOI: 10.1145/192161.192286. URL: https://dl.acm.org/doi/10.1145/192161.192286 (visited on 09/11/2025).

[106] Paul Westermann and Ralph Evins. “Surrogate modelling for sustainable building design – A review”. In: *Energy and Buildings* 198 (Sept. 2019), pp. 170–186. ISSN: 0378-7788. DOI: 10.1016/j.enbuild.2019.05.057. URL: https://www.sciencedirect.com/science/article/pii/S0378778819302877 (visited on 09/11/2025).

[107] E. Zitzler and L. Thiele. “Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach”. In: *IEEE Transactions on Evolutionary Computation* 3.4 (Nov. 1999), pp. 257–271. ISSN: 1941-0026. DOI: 10.1109/4235.797969. URL: https://ieeexplore.ieee.org/document/797969 (visited on 09/11/2025).

List of Author's Contributions

[1] "Adaptive Hyperparameter Optimization for Constraint Solvers using the Probe and Solve Algorithm (PSA)". Talk at University of Luxembourg. 2024.

[2] "Building Information Modelling (BIM) and its integration with Artificial Intelligence (AI)". Talk at ILNAS Journée mondiale de la normalisation (World Standards Day). 2022. URL: https://portail-qualite.public.lu/fr/actualites/normes-normalisation/2022/journee-mondiale-normalisation-2022-grand-duche-luxembourg.html.

[3] Haddad, Hedieh and Talbot, Pierre and Bouvry, Pascal. "Selecting Search Strategy in Constraint Solvers using Bayesian Optimization". In: *IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI 2024)*. IEEE. 2024, pp. 764–773.

[4] Hedieh Haddad, Pierre Talbot, Pascal Bouvry. "Comparison of Hyperparameter Optimization Methods for Selecting Search Strategy of Constraint Programming Solvers". In: *PTHG-24: The Seventh Workshop on Progress Towards the Holy Grail*. 2024. URL: https://freuder.wordpress.com/progress-towards-the-holy-grail-workshops/pthg-24-the-seventh-workshop-on-progress-towards-the-holy-grail/.

[5] Jean-Philippe Humbert et al. *Research-driven Standardization Opportunities for ICT, Construction and Aerospace*. Institut luxembourgeois de la normalisation, de l'accréditation, de la sécurité et qualité des produits et services (ILNAS), 2024.

[6] "Hyperparameter Optimization of Constraint Programming Solvers". Talk at University of Luxembourg. 2023.

[7] "Presentation of construction section of the white paper "Research-driven Standardization Opportunities for ICT, Construction and Aerospace"". Talk at ILNAS Journée mondiale de la normalisation (World Standards Day). 2024. URL: https://portail-qualite.public.lu/fr/actualites/normes-normalisation/2024/journee-mondiale-normalisation-2024-moment-echanges-normalisation-luxembourg.html.

[8] "Presentation of construction section of the white paper "Trustworthiness in ICT, aerospace and construction applications – Scientific research and technical standardization"". Talk at ILNAS Journée mondiale de la normalisation (World Standards Day). 2023. URL: https://portail-qualite.public.lu/fr/actualites/normes-normalisation/2023/invitation-journee-mondiale-normalisation-2023.html.

[9] Jean-Marie Reiff et al. *Trustworthiness in ICT, Aerospace and Construction Applications*. Institut luxembourgeois de la normalisation, de l'accréditation, de la sécurité et qualité des produits et services (ILNAS), 2023.

[10] *Technical Standardisation for Trustworthy ICT, Aerospace and Construction*. SnT Partnership Day, European Convention Center Luxembourg. 2022.

[11] *Technical Standardisation for Trustworthy ICT, Aerospace and Construction*. SnT Partnership Day, European Convention Center Luxembourg. 2023.

[12] *Technical Standardisation for Trustworthy ICT, Aerospace and Construction*. SnT Partnership Day, European Convention Center Luxembourg. 2024.