mathematics

MDPI

Article

A Comprehensive Benchmark of Constraint Programming Solvers for the Makespan-Minimisation Job Shop Scheduling Problem

Francisco Yuraszeck [1], Frank Werner [2,*] and Daniel Rossit [3,*]

[1] Department of Engineering Sciences, Universidad Andres Bello, Quillota 980, Viña del Mar 2531015, Chile; francisco.yuraszeck@unab.cl
[2] Faculty of Mathematics, Otto-von-Guericke-Universität Magdeburg, 39110 Magdeburg, Germany
[3] Departamento de Ingeniería, Universidad Nacional del Sur, INMABB, CONICET, Bahía Blanca 8000, Argentina
* Correspondence: frank.werner@ovgu.de (F.W.); daniel.rossit@uns.edu.ar (D.R.)

Abstract

The job shop scheduling problem (JSSP) is a paradigmatic and strongly $\mathcal{NP}$-hard combinatorial optimisation problem that underpins production planning in modern manufacturing systems, and constraint programming (CP) has become one of the leading methodologies for tackling it. However, comparative studies of CP solvers for the JSSP have so far been restricted to a single benchmark family, a single instance-size range, or a single hardware setting, which limits the practical guidance they offer to both researchers and practitioners. This paper presents a controlled empirical evaluation of four state-of-the-art CP solvers—IBM ILOG CP Optimizer, Google OR-Tools (CP-SAT), Hexaly, and OptalCP—on the makespan-minimisation JSSP. The four engines are run with default parameters and a uniform 600 s wall-clock time budget on 332 instances drawn from nine canonical benchmark families (Fisher–Thompson, Lawrence, Adams–Balas–Zawack, Applegate–Cook, Yamada–Nakano, Storer–Wu–Vaccari, Taillard, Demirkol–Mehta–Uzsoy, and Da Col–Teppan), spanning sizes from 6×6 to 1000×1000 operations. OptalCP emerges as the most robust engine overall, certifying optimality on 191 of the 332 instances (57.5%) with the smallest average optimality gap (3.55%), followed by CP Optimizer (166 optima), OR-Tools (144), and Hexaly (116), while Hexaly dominates on industrial-scale problems and produces the bulk of the 22 new best-known upper bounds and one new best-known lower bound reported here. A Friedman test followed by Nemenyi post hoc comparisons confirms that OptalCP attains significantly smaller optimality gaps than the three other engines ($p < 0.001$). Solver competitiveness depends sharply on instance size and the n/m ratio, with square instances confirmed as the hardest case. In practical terms, these findings support an instance-aware approach to CP solver selection: OptalCP is the default choice for small to large instances of moderate aspect ratio, whereas Hexaly is preferable for industrial-scale problems with tens of thousands of operations or extreme n/m ratios, where it is the only engine that reliably returns high-quality feasible schedules within the time budget.

Keywords: job shop scheduling; constraint programming; benchmarking; CP Optimizer; OR-Tools; CP-SAT; Hexaly; OptalCP; makespan minimisation; industrial-scale scheduling

MSC: 68M20; 68W40

Check for updates

Academic Editor: Andrea Scozzari

Received: 18 May 2026
Revised: 8 June 2026
Accepted: 12 June 2026
Published: 17 June 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Production scheduling is one of the cornerstones of modern operations management. By determining the timing and assignment of operations to limited resources, scheduling translates strategic and tactical plans into executable decisions that govern shop-floor performance, lead times, work-in-process inventory, and customer service [1,2]. The economic relevance of scheduling has only increased with the diffusion of digital manufacturing paradigms such as Industry 4.0 and the Smart Factory, in which the planning and control loops are tightened and decision-support systems are expected to deliver high-quality schedules within strict time budgets [3–5]. In this context, optimisation methods are no longer a niche tool reserved for academic studies: they form the algorithmic backbone of advanced planning and scheduling software, manufacturing execution systems, and emerging cyber–physical platforms.

Among the wide variety of scheduling problems, the job shop scheduling problem (JSSP) plays a paradigmatic role. The JSSP captures a setting in which a set of jobs, each composed of a fixed sequence of operations, must be processed on heterogeneous machines that can handle one operation at a time [1,6,7]. Despite its seemingly compact definition, the JSSP is strongly $\mathcal{NP}$-hard already in its makespan-minimisation variant $J \,||\, C_{\max}$ [8], and small instances are notoriously difficult to solve to proven optimality. Beyond its theoretical interest, the JSSP is widely regarded as a canonical abstraction of discrete manufacturing environments, encompassing features such as heterogeneous resources, precedence constraints, sequence-dependent processing routes, and tight resource contention [9–11]. For these reasons, it is routinely used as a testbed for new optimisation paradigms, and progress on the JSSP has historically driven methodological innovation that later propagated to flow shops, flexible job shops, open shops, and resource-constrained project scheduling [12,13].

Within the spectrum of optimisation paradigms applied to scheduling, constraint programming (CP) has evolved from a relatively niche academic technology into a leading methodology for solving real-world scheduling problems [12,14,15]. CP's declarative modelling style, in which constraints express relations among decision variables rather than procedural assignment rules, aligns naturally with scheduling concepts such as precedences, machine capacities, time windows, calendars, and renewable or cumulative resources. Modern CP solvers exploit specialised global constraints (e.g., `noOverlap`, `cumulative`, `disjunctive`, and `alternative`) and powerful search procedures, including failure-directed search [16], large-neighbourhood search, lazy clause generation, and reinforcement-learning-guided heuristics [17]. As a result, the CP technology now competes with, and often surpasses, mixed-integer linear programming (MILP) on a variety of scheduling benchmarks, both in terms of solution quality and bound strength [18,19].

Several mature CP platforms are available to practitioners and researchers. IBM ILOG CP Optimizer offers a robust "model-and-run" framework with specialised support for interval variables, cumul functions, and complex objectives such as earliness, tardiness, and non-execution costs [15]. Google OR-Tools provides the open-source CP-SAT solver, which combines lazy clause generation with linear programming relaxations and has consistently won the international MiniZinc Challenge since 2013 [20,21]. Hexaly (formerly LocalSolver) offers an alternative paradigm based on compact modelling with interval and list variables, lambda expressions, and a hybrid metaheuristic engine designed for industrial-scale instances of up to one million activities [22,23]. More recently, OptalCP [24] has emerged as a new entrant in the CP-for-scheduling ecosystem: developed by ScheduleOpt and led by Petr Vilím, the original architect of the search algorithms behind CP Optimizer, OptalCP combines an interval-based scheduling vocabulary similar to that of CP Optimizer with a parallel-search portfolio that integrates failure-directed search variants, large-neighbourhood search, asynchronously supplied user heuristics, and

reinforcement-learning-based acceleration mechanisms [16,17]; it provides native APIs (application programming interfaces) in Python and JavaScript/TypeScript, which facilitates its integration into modern decision-support pipelines, and is freely available for academic and evaluation use. Each of these solvers implements distinct propagation engines, search heuristics, and modelling idioms, with the consequence that their relative performance is highly instance-dependent: some platforms excel at finding feasible solutions quickly, while others prove optimality more often or produce stronger lower bounds. Empirical evidence collected on canonical instance sets has consistently demonstrated this heterogeneity [17,25].

The architectural and implementation differences across CP solvers give rise to substantial performance variation across instance types, problem sizes, and objective formulations. Consequently, systematic benchmarking under controlled conditions has become essential to characterise solver capabilities, expose strengths and weaknesses, and guide both methodological design and tool selection in industrial deployments. Prior studies have shown that CP solvers exhibit highly disparate behaviour across benchmark families, instance sizes, and objective formulations [18,25,26], motivating a broader and more systematic empirical evaluation.

Building on these insights, the present paper extends our previous benchmarking effort along three complementary axes. First, the empirical evaluation is significantly broadened: rather than the 80 instances of the Taillard set, we evaluate the four solvers on 332 instances drawn from nine well-established benchmark families: Fisher and Thompson, Lawrence, Adams–Balas–Zawack, Applegate–Cook, Yamada–Nakano, Storer–Wu–Vaccari, Taillard, Demirkol–Mehta–Uzsoy, and the recent Da Col and Teppan large-scale instances [19,27]. This collection includes both classical small instances that have been used as standard testbeds for decades and modern hard instance generators specifically designed to challenge state-of-the-art exact and heuristic methods. Second, we explicitly target large and realistic problem sizes by including the Da Col–Teppan industrial-style instances, which scale up to 1000 jobs and 1000 machines and represent realistic abstractions of large-scale manufacturing environments, such as semiconductor fabrication and complex assembly lines [19,27]. To the best of our knowledge, no prior comparative study reports CP solver performance across this full size range under identical hardware and time-budget conditions. Third, the analysis goes beyond aggregate averages and examines per-family behaviour, the evolution of optimality gap, upper-bound quality ($\mathrm{RPD}_{\mathrm{UB}}$), lower-bound quality ($\mathrm{RPD}_{\mathrm{LB}}$), and runtime distributions as instance size grows.

The contributions of this paper can therefore be summarised as follows. (i) We present the largest, to the best of our knowledge, controlled benchmarking of four modern CP solvers on the makespan-minimisation JSSP, encompassing 332 instances from nine canonical benchmark sets and covering a range of sizes from 6×6 to 1000×1000. (ii) We document and analyse the relative behaviour of CP Optimizer, OR-Tools (CP-SAT), Hexaly and OptalCP under identical hardware and a uniform time budget, providing fine-grained insights into how each solver scales with instance size and benchmark structure. (iii) We assess the practical implications of our findings for practitioners deploying CP technology in industrial environments, in particular regarding solver selection, expected upper- and lower-bound quality, and the regions of the parameter space in which each engine appears to be most competitive. (iv) We discuss methodological lessons for the empirical evaluation of CP solvers in scheduling, including the importance of instance diversity, the role of bound quality alongside primal performance, and the interaction between solver design choices and problem structure.

In line with these contributions, the present study is guided by four research questions. (RQ1) How do the four CP solvers compare in terms of proven optimality, optimality

gap, and primal- and dual-bound quality on a large and heterogeneous collection of JSSP instances run under identical hardware and time-budget conditions? (RQ2) How does the relative competitiveness of the solvers evolve as a function of instance size and the n/m aspect ratio? (RQ3) To what extent can off-the-shelf CP solvers, used with default parameters, improve the best-known bounds for industrial-size instances? (RQ4) What practical guidance can be distilled for practitioners who must select a CP engine for a given scheduling application? To position our study with respect to earlier benchmarking efforts, Table 1 summarises the scope of representative prior comparisons along the dimensions that matter for solver evaluation: the number and recency of the engines considered, the number of benchmark families, the largest instance size reported, whether a single hardware platform and a uniform time budget were used, and whether the comparison was supported by a formal statistical analysis.

Table 1. Positioning of this study with respect to representative prior benchmarking works on CP (and CP versus MILP) for the JSSP and related shop-scheduling problems. "Families" denotes the number of distinct benchmark instance families considered; "max. size" is the largest instance size reported ($n \times m$); "same HW/time" indicates whether all engines were run on identical hardware under a uniform time budget; "stat. test" indicates whether a formal statistical comparison of the solvers was reported.

Study	Solvers Compared	Families	Max. Size	Same HW/Time	Stat. Test
Beck et al. [28]	CP, local search (2)	4	50×10	Yes	No
Da Col & Teppan [19]	CP Optimizer (1)	2	1000×1000	Yes	No
Müller et al. [25]	CP Optimizer, OR-Tools (2)	1 (FJSSP)	20×15	Yes	No
Naderi et al. [18]	CP Optimizer, MILP (2)	several	100×20	Yes	No
Conference version [26]	CP Optimizer, OR-Tools, Hexaly (3)	1 (Taillard)	100×20	Yes	No
This study	**CP Optimizer, OR-Tools, Hexaly, OptalCP (4)**	**9**	**1000×1000**	**Yes**	**Yes**

The remainder of the article is organised as follows. Section 2 reviews the literature on the JSSP and CP for scheduling, with an emphasis on solving methods, lower-bounding techniques, and the benchmark sets used in this work. Section 3 formally introduces the $J \,||\, C_{\max}$ problem and describes the CP formulations, the experimental protocol, and the performance indicators. Section 4 reports and discusses the results across the nine benchmark families, including aggregated and per-family analyses. Section 5 discusses the broader implications of our findings. Finally, Section 6 concludes and outlines avenues for future research.

2. Related Work

This section reviews the literature that is relevant to our study. We begin by recalling the formal nature of the JSSP and the historical evolution of solution methods (Section 2.1). We then focus on constraint programming approaches and the modern CP solvers used in this work (Section 2.2). Next, we discuss empirical comparisons and benchmarking studies of CP solvers, with an emphasis on those targeting the JSSP and large-scale or industrial scheduling problems (Section 2.3). Finally, we review the canonical benchmark instance families used in our experiments and motivate their selection (Section 2.4).

2.1. The Job Shop Scheduling Problem

The classical JSSP is formally defined over a set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ to be processed on a set of m machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job J_j consists of an ordered sequence of m operations $O_{j,1}, O_{j,2}, \ldots, O_{j,m}$, where operation $O_{j,k}$ has to be processed on a specific machine $\mu(j,k) \in \mathcal{M}$ for a deterministic processing time $p_{j,k}$. Each machine can

process at most one operation at a time, operations are non-preemptive, and the order of operations within each job must be respected. The objective considered in this work, denoted $J \,||\, C_{\max}$ in the standard $\alpha \mid \beta \mid \gamma$ notation [29], is to minimise the makespan, that is, the completion time of the last operation. The decision version of the JSSP is strongly $\mathcal{NP}$-hard [8], and even special cases with three machines or with very limited routing patterns remain intractable in the worst case [30].

Research on the JSSP spans more than six decades, and an exhaustive review is beyond the scope of this work. We therefore focus on the most relevant methodological strands to our benchmarking effort: exact methods, metaheuristics, and lower-bounding procedures.

Exact methods. The earliest mathematical-programming formulations of the JSSP date back to Manne's disjunctive MILP model [31] and Wagner's alternative formulation [32], which together established the foundations of the disjunctive-graph view of the problem. Subsequent decades witnessed the development of branch-and-bound algorithms exploiting the disjunctive graph and edge-finding techniques [33,34], time-window shaving [35], and dynamic-programming approaches that exploit specific structural properties [36]. More recently, SAT encodings of the JSSP and its open-shop sister problem have proven to be competitive on selected instances [37,38]. Despite this progress, exact methods continue to struggle on instances of moderate to large size, motivating the development of specialised metaheuristics and the use of CP technology, which is the focus of our work.

Metaheuristics and hybrid methods. The metaheuristic literature on the JSSP is vast. Tabu search algorithms have been particularly successful, with the i-TSAB algorithm by Nowicki and Smutnicki [39] long considered a state-of-the-art reference. Other prominent contributions include simulated annealing [40], global equilibrium search [41], biased random-key genetic algorithms with local search [42], path-relinking and hybrid GA–TS combinations [9], and large-neighbourhood search procedures embedded in commercial CP solvers. Recent surveys provide a comprehensive overview of the modern landscape [9,10]. While these methods often produce best-known upper bounds on classical instance sets, they typically do not provide optimality certificates, which has shifted attention towards exact and bound-strengthening procedures within CP.

Lower-bounding procedures. Strengthening lower bounds is a central concern for the JSSP, both because tight bounds reduce the optimality gap and because they help to certify the quality of heuristic solutions. Classical analytical bounds based on machine and job loads were enhanced by shaving [35] and time-window tightening [34]. More recent contributions include translations of the JSSP into resource-constrained project scheduling problem (RCPSP) relaxations, which can then be solved with specialised CP-based bounding procedures [43,44], as well as machine-learning-based lower-bounding procedures that learn to predict tight bounds from instance features [45]. Failure-directed search [16] and propagation-centric algorithms have similarly produced tighter lower bounds on hard benchmark sets. Empirical comparisons increasingly highlight the value of CP-based bounds, and our experiments quantify how this advantage translates to practice across multiple instance families.

2.2. Constraint Programming for Scheduling

Constraint programming has emerged as a particularly influential paradigm for scheduling problems. CP models reason about activities and resources via specialised global constraints that express disjunctive resource usage, cumulative resources, alternative machines, and synchronisation. The combination of expressive modelling primitives with strong propagation algorithms makes CP especially attractive for combinatorial scheduling problems with rich side constraints [12,14,15]. A growing body of empirical evidence shows that CP can outperform MILP on scheduling benchmarks [17–19] and that

CP technology can be successfully embedded in production-control loops and decision-support systems [12,19].

Three of the most widely used CP platforms are CP Optimizer, Google OR-Tools (CP-SAT), and Hexaly. We briefly review their main features since they constitute the solvers under analysis in this work.

IBM ILOG CP Optimizer. CP Optimizer [15] is a commercial CP system specifically designed for combinatorial optimisation and scheduling. It exposes a high-level modelling language based on interval variables, optional intervals, sequence variables, and cumul functions. CP Optimizer's search engine relies on failure-directed search [16] together with restart-based large-neighbourhood search and learning mechanisms. The solver is intended to follow a "model-and-run" philosophy that minimises the need for manual parameter tuning, and it has been used in a wide range of applications, including detailed scheduling, routing, and resource allocation. Empirical studies have repeatedly identified CP Optimizer as one of the strongest off-the-shelf solvers for scheduling problems [12,18,25].

Google OR-Tools (CP-SAT). OR-Tools is an open-source optimisation suite developed by Google [20], whose flagship CP component is the CP-SAT solver [21]. CP-SAT integrates Boolean SAT reasoning, lazy clause generation, linear-programming relaxations, portfolio search with multiple parallel workers, and global constraints tailored for scheduling, such as `noOverlap` and `cumulative`. CP-SAT has consistently dominated the international MiniZinc Challenge since 2013 and has been extensively used both in research and industrial applications. Its open-source nature has also contributed to its widespread adoption as a research baseline [17,18].

Hexaly. Hexaly (formerly LocalSolver) [22,23] provides a hybrid optimisation paradigm that combines a compact expressive modelling language based on interval and list variables with a metaheuristic engine that exploits problem structure to perform large-neighbourhood moves. Hexaly's design specifically targets industrial-scale instances and has demonstrated the ability to find high-quality feasible solutions for problems with up to one million activities. The solver supports major programming interfaces, including Python 3.14.6., Java 26.0.1, C# 14, and C++ 23, and recent applications report competitive performance on a variety of scheduling and routing problems [22,23].

Beyond these three engines, additional CP platforms relevant to scheduling include Choco [46], Gecode [47], the MiniZinc ecosystem [48], and recent learning-augmented CP frameworks that incorporate reinforcement-learning-based branching policies and learned heuristics [17]. While these tools were not included in our experiments, their existence underscores the diversity of the CP solver landscape and the difficulty of drawing universal conclusions from any single benchmarking study.

2.3. Empirical Comparisons and Benchmarking Studies

Empirical benchmarking of CP solvers and competing optimisation paradigms has become an established research activity. Naderi, Ruiz, and Roshanaei [18] compared MILP and CP formulations on a wide range of shop scheduling problems, concluding that CP formulations often dominate MILP both on solution quality and bound strength. Da Col and Teppan [19] reported on the use of CP to tackle industrial-size JSSP instances, introducing a generator of large-scale scheduling problems and showing that CP-based methods can produce high-quality solutions even for instances with thousands of jobs and machines. Müller et al. [25] studied algorithm-selection strategies for the flexible job shop scheduling problem and proposed a machine-learning-based approach to choose between CP solvers. Heinz, Vilím, and Hanzálek [17] demonstrated how reinforcement learning can reduce search-tree size in CP-based scheduling, with a positive impact on both job shop and resource-constrained project scheduling benchmarks. Prata, Abreu, and Nagano [12]

provide a recent bibliometric survey of CP applications to production scheduling, evidencing the growing prominence of CP in the area. Earlier benchmarking efforts, such as those of Beck [49], Beck, Feng, and Watson [28], and Vilím, Laborie, and Shaw [16], established methodological foundations and laid the groundwork for the systematic evaluation of modern CP solvers.

Three lines of work are particularly relevant to the way CP is deployed in current scheduling practice and frame the contribution of the present study. The first line is the *hybridisation of CP with mathematical programming*. Modern CP-SAT-style engines already integrate linear-programming relaxations and Boolean reasoning inside the search [21], and head-to-head comparisons of CP against MILP on a wide range of shop-scheduling problems show that the two paradigms are complementary, with CP typically dominating on disjunctive scheduling and MILP remaining competitive on more loosely constrained variants [18,50]. The second line is *machine-learning-assisted CP*, in which learned policies guide branching, restart, or bounding decisions: Heinz, Vilím, and Hanzálek [17] show that reinforcement learning can substantially reduce the search-tree size of a CP scheduler, and learning-based lower-bounding procedures [45] predict tight bounds from instance features. The third line is *portfolio-based and per-instance solver selection*, where the goal is not to crown a single best engine but to choose, for each instance, the most promising solver from a portfolio; Müller et al. [25] demonstrate the value of this approach for the flexible job shop, and hybrid CP–local-search portfolios have long been known to outperform any single component [28]. Against this backdrop, the novelty of the present study is threefold: (i) it is, to the best of our knowledge, the first comparison to include OptalCP alongside the three established engines; (ii) it spans nine benchmark families and four orders of magnitude of instance size under strictly identical hardware and time budgets, a breadth not matched by previous CP benchmarking works (see Table 1); and (iii) it backs its ranking with a formal statistical analysis, providing the per-instance, per-size, and per-n/m-ratio evidence that portfolio and algorithm-selection methods require as input.

Despite this rich literature, several gaps persist. First, most comparative studies are restricted to a single benchmark family, often Taillard [51] or Demirkol–Mehta–Uzsoy [52], which may bias conclusions about solver behaviour. Second, the assessment of solver performance under identical hardware, identical time budgets, and identical default-parameter settings is uncommon, making cross-paper comparisons difficult. Third, very few studies analyse how solvers scale to truly industrial-size instances; Da Col and Teppan's seminal contribution [19,27] is among the few notable exceptions. The previous conference paper [26] addressed the first two gaps for the Taillard set but did not include large-scale instances and did not compare across the diverse benchmark families used in the metaheuristic and operations-research literature. The present work directly addresses these gaps.

2.4. Benchmark Instance Families

The benchmarking experiments reported in this paper cover nine well-established families of JSSP instances, totalling 332 problems, presented by $n \times m$. We summarise the main features of each family below.

Fisher and Thompson (FT). The Fisher and Thompson instances [53], including the famous `ft06`, `ft10`, and `ft20` benchmarks, are among the oldest and most-cited JSSP testbeds. They have served as standard references for decades. While small in size, they are widely used as sanity checks for new methods.

Lawrence (LA). The 40 Lawrence instances [54] cover sizes ranging from 10×5 to 30×10 and have been a staple of the JSSP literature for over four decades. Their relatively

diverse size combinations and difficulty levels make them a useful reference set for general-purpose comparisons.

Adams, Balas, and Zawack (ABZ). The five ABZ instances [55] were introduced in the seminal paper proposing the shifting-bottleneck procedure. Sizes range from 10×10 to 20×15, and instances such as `abz7`–`abz9` are still considered challenging for many heuristic and exact methods.

Storer, Wu, and Vaccari (SWV). The 20 SWV instances [56] cover sizes from 20×10 to 50×10 and were specifically designed to challenge constructive heuristics; many of them remain open in the metaheuristic literature.

Yamada and Nakano (YN). The four 20×20 Yamada and Nakano instances [57] are renowned for their difficulty and have been used to evaluate genetic algorithms, simulated annealing, and recent CP-based methods.

Applegate and Cook (ORB). The ORB instances [58] are a set of 10 carefully crafted 10×10 problems originally introduced as part of an early branch-and-cut study. Despite their small size, several of them are non-trivial for CP-based exact methods.

Taillard. The 80 Taillard instances [51] have become the de facto standard benchmark for the JSSP. They span sizes from 15×15 to 100×20 and have been used to evaluate virtually every JSSP solution method proposed in the last three decades. Several Taillard instances remain open, with their best-known lower bounds recently improved by CP-based bounding procedures [43,44].

Demirkol, Mehta, and Uzsoy (DMU). The 80 DMU instances [52] were specifically designed to be harder than Taillard's. Sizes range from 20×15 to 50×20, and many DMU instances are routinely used to evaluate state-of-the-art metaheuristics for the JSSP.

Da Col and Teppan (DCT). The Da Col and Teppan generator [19,27] produces large-scale JSSP instances aimed at industrial applications. The set of 90 DCT instances used in this work covers sizes from 10×10 to 1000×1000, providing the largest benchmark family considered. Their inclusion is one of the distinguishing features of our study as it allows us to characterise CP solver behaviour at scales seldom reported in the literature.

Together, these nine families span more than four orders of magnitude in instance size and a wide variety of structural properties (load profiles, job-to-machine ratios, and generation processes), making them a representative testbed for empirically characterising the behaviour of contemporary CP solvers on the JSSP.

3. Problem and Solution Approach

This section formalises the problem under study (Section 3.1), introduces a basic constraint programming (CP) model for the makespan-minimisation job shop scheduling problem (JSSP) that explicitly captures the decisions and resource requirements involved (Section 3.2), and describes the four CP-based optimisation engines used in our experiments: IBM ILOG CP Optimizer, Google OR-Tools (CP-SAT), Hexaly, and OptalCP (Section 3.3).

3.1. Problem Statement

Let $\mathcal{J} = \{1, 2, \ldots, n\}$ be a set of jobs to be processed on a set of machines $\mathcal{M} = \{1, 2, \ldots, m\}$. Each job $j \in \mathcal{J}$ consists of an ordered sequence of m operations $O_j = (O_{j,1}, O_{j,2}, \ldots, O_{j,m})$ such that operation $O_{j,k}$ must be processed on a fixed machine $\mu(j,k) \in \mathcal{M}$ for a fixed processing time $p_{j,k} \in \mathbb{Z}_{>0}$. The mapping $\mu(j, \cdot)$ is a permutation of $\mathcal{M}$, so each job visits every machine exactly once. The following assumptions hold: (i) every machine processes at most one operation at a time; (ii) operations are non-preemptive; (iii) the order of operations within each job must be respected; (iv) machines are continuously available from time zero; and (v) no setup or transportation times are explicitly modelled since they are assumed to be embedded in the processing times.

Let $S_{j,k}$ denote the start time of $O_{j,k}$ and $C_{j,k} = S_{j,k} + p_{j,k}$ its completion time. The makespan is defined as $C_{\max} = \max_{j \in \mathcal{J}} C_{j,m}$. The decision problem $J \,||\, C_{\max}$ asks whether a feasible schedule with makespan no greater than a given threshold exists; the corresponding optimisation problem seeks a feasible schedule of minimum makespan and is strongly $\mathcal{NP}$-hard [8]. We denote a problem instance compactly as $n \times m$.

For ease of reference, the notation used throughout the paper is the following. The *sets* are the jobs $\mathcal{J} = \{1, \ldots, n\}$ and the machines $\mathcal{M} = \{1, \ldots, m\}$; the *indices* j and i range over jobs and machines, respectively, and $k \in \{1, \ldots, m\}$ indexes the position of an operation within a job. The *parameters* are the routing map $\mu(j,k) \in \mathcal{M}$, which assigns operation $O_{j,k}$ to a machine, and the processing time $p_{j,k} \in \mathbb{Z}_{>0}$; the horizon $H = \sum_{j,k} p_{j,k}$ is a derived parameter. The *decision variables* are the operation start times $S_{j,k} \geq 0$, from which the completion times $C_{j,k} = S_{j,k} + p_{j,k}$ and the objective $C_{\max} = \max_j C_{j,m}$ follow. The *constraints* are the per-job precedence relations and the per-machine disjunctive (no-overlap) relations, together with the non-preemption and unit-capacity assumptions (i)–(v) stated above. The strong $\mathcal{NP}$-hardness of $J \,||\, C_{\max}$ is not merely a worst-case artefact: it holds already for $m = 3$ machines [8,30], no polynomial-time constant-factor approximation is known for arbitrary m, and even instances with as few as 10×10 operations (such as the celebrated `ft10`) resisted exact resolution for more than two decades. This intrinsic difficulty is precisely what motivates the use of constraint programming, whose strong propagation and bounding mechanisms are designed to cope with the combinatorial explosion that exact enumeration would otherwise face.

3.2. A Basic CP Model for the JSSP

Constraint programming represents scheduling decisions through interval variables, sequence variables, and global constraints that propagate temporal and resource information [14,15]. We present below a basic CP model for the $J \,||\, C_{\max}$ problem that constitutes the conceptual backbone of all four solver-specific implementations described in Section 3.3. The model uses the modelling vocabulary established by IBM ILOG CP Optimizer [15] and adopted, with minor syntactic differences, by Google OR-Tools, Hexaly, and OptalCP.

Decision variables. For every operation $O_{j,k}$ we define an *interval variable* $I_{j,k}$ with fixed length $p_{j,k}$:

$$I_{j,k} = \text{intervalVar}(size = p_{j,k}), \qquad \forall j \in \mathcal{J},\ k \in \{1, \ldots, m\}. \tag{1}$$

Interval variables encapsulate the start time, end time, and length of an operation in a single declarative object, allowing CP solvers to reason about them through specialised propagation algorithms. We will write $startOf(I)$, $endOf(I)$, and $lengthOf(I)$ for the corresponding numerical accessors.

Precedence constraints. For each job, the operations must be processed in the prescribed order:

$$endOf(I_{j,k}) \leq startOf(I_{j,k+1}), \qquad \forall j \in \mathcal{J},\ k \in \{1, \ldots, m-1\}. \tag{2}$$

This precedence relation is typically enforced through the global constraint `endBeforeStart` (or its equivalent in OR-Tools and Hexaly), which provides stronger propagation than a pairwise inequality.

Resource constraints. For each machine $i \in \mathcal{M}$, the set of operations that must be processed on i is

$$\mathcal{O}_i = \{I_{j,k} : \mu(j,k) = i\}, \qquad \forall i \in \mathcal{M}. \tag{3}$$

The disjunctive nature of each machine—at most one operation in process at any time—is captured by the global constraint `noOverlap`:

$$\text{noOverlap}(\mathcal{O}_i), \qquad \forall i \in \mathcal{M}. \tag{4}$$

The `noOverlap` constraint encapsulates classical edge-finding, not-first/not-last, and detectable-precedence propagation, which together prune dominated assignments and tighten the time bounds of the operations on the machine [14,16].

Time origin and bounds. Without loss of generality, the time horizon is set to $H = \sum_{j,k} p_{j,k}$, which is a trivial upper bound on the makespan; tighter horizons are obtained from constructive heuristics or known upper bounds. The starting time of every operation is therefore restricted to $[0, H - p_{j,k}]$.

Objective function. The makespan is the maximum end time across the last operation of each job:

$$\min \; C_{\max} = \max_{j \in \mathcal{J}} \mathit{endOf}(I_{j,m}). \tag{5}$$

Modern CP solvers expose this objective as a `makespan` expression that is integrated into their branch-and-bound and dual-bound machinery.

Therefore, the full model is (1)–(5). It is a compact and faithful CP encoding of the $J \,\|\, C_{\max}$ problem; equivalent formulations have been used in the seminal contributions on CP-based scheduling [14–16] and in subsequent comparative studies [18,19]. The four solvers used in our experiments differ primarily in the search strategy, the propagation engines, and the parallelisation paradigm employed to explore the search space rather than in the model formulation itself.

It is worth emphasising that the choice of global constraints, the propagation level, the search and branching heuristics, and any symmetry-handling mechanism are, in every case, those provided by each solver in its default configuration. Each of these aspects is known to have a substantial impact on performance, and a dedicated tuning of, for instance, the no-overlap propagation strength or an explicit symmetry-breaking scheme could in principle improve the results of any individual engine. However, deliberately exercising such solver-specific adaptations would compromise the out-of-the-box model-and-run comparison that is the object of this study (see Section 3.4); we therefore keep the model and all search-related settings at their default values across the four solvers and leave a systematic investigation of these enhancements as future work.

3.3. Constraint Programming Solvers

We now describe the four CP solvers compared in this work. The first three coincide with the engines benchmarked in the conference version of this study [26]; the fourth, OptalCP, is incorporated here for the first time and represents a recent and increasingly relevant addition to the CP-for-scheduling ecosystem.

3.3.1. IBM ILOG CP Optimizer

CP Optimizer [15] is the commercial CP system developed by IBM as part of the ILOG CPLEX Optimization Studio. It provides a dedicated modelling layer for scheduling—with first-class support for interval variables, optional intervals, sequence variables, cumulative functions, and state functions—and a search engine designed to operate competitively with minimal user tuning (the so-called "model-and-run" philosophy). Its automatic search relies on failure-directed search (FDS) [16], large-neighbourhood search, learning, and restart mechanisms operating in parallel across multiple worker threads. CP Optimizer is widely regarded as a strong reference baseline for benchmarking CP-based scheduling

solvers [12,18,25]. In our experiments we use CP Optimizer 22.1.2 and the JSSP formulation provided in the `SchedJobShop.java26.0.1` example shipped with the software.

3.3.2. Google OR-Tools (CP-SAT)

OR-Tools is an open-source optimisation suite developed by Google [20]. Its flagship CP component, the CP-SAT solver [21], integrates Boolean SAT reasoning, lazy clause generation, linear-programming relaxations, portfolio search across heterogeneous workers, and global constraints tailored for scheduling, such as `noOverlap` and `cumulative`. CP-SAT has consistently won the international MiniZinc Challenge since 2013, and its open-source nature makes it a popular research baseline. In our experiments we use OR-Tools v9.14.6206 and the JSSP formulation provided on the official documentation page.

3.3.3. Hexaly

Hexaly (formerly LocalSolver) [22,23] adopts a fundamentally different paradigm. Its modelling language is built around compact *interval* and *list variables* together with lambda expressions, which allow scheduling problems to be described with very few lines of code. Internally, Hexaly relies on a hybrid metaheuristic engine that combines aggressive moves on list variables with mathematical-programming-style local consistency checks. Its design is explicitly tailored to industrial-scale instances, and the solver has demonstrated the ability to find high-quality feasible solutions on problems with up to one million activities. We use Hexaly 14.0 and the JSSP formulation distributed in the official Hexaly examples.

3.3.4. OptalCP

OptalCP (https://optalcp.com (accessed on 5 February 2026)) [24] is a recent CP solver developed by ScheduleOpt (CoEnzyme SAS 2026) and led by Petr Vilím, the original designer of the search algorithms behind IBM ILOG CP Optimizer. Like CP Optimizer, it adopts an interval-based scheduling vocabulary and provides high-level constructs, such as `noOverlap`, `cumulative`, `alternative`, and `stepFunction`. OptalCP is distinguished by three design choices: (i) its native APIs target Python and JavaScript/TypeScript 26.0.1 through the package `@scheduleopt/optalcp`, which makes it easy to embed in modern decision-support pipelines; (ii) its parallel search combines a portfolio of failure-directed search variants [16], large-neighbourhood search, and asynchronously supplied user heuristics that can inject improving solutions at runtime; and (iii) the solver incorporates reinforcement-learning-based acceleration mechanisms—in particular a multi-armed bandit-style controller—to adaptively bias search-tree exploration towards productive regions of the space [17]. OptalCP is freely available for academic and evaluation use, and recent comparisons report competitive performance against CP Optimizer, CPLEX, and Gurobi on multiple scheduling benchmarks [24]. In our experiments we use OptalCP 2026.2.0.

3.4. Experimental Protocol

All four solvers use their default parameter settings, except for an enforced *wall-clock* time limit of 600 s per instance. Each solver is allowed to use all the threads available on the machine. A single hardware platform is used for all the experiments, ensuring strictly comparable runtime measurements. All the instances are loaded from the original benchmark text files; for each solver we use the JSSP modelling formulation distributed in its official example directory so as to evaluate each engine in conditions representative of the experience of a non-expert practitioner.

To make the protocol fully reproducible, we detail the following points. No solver parameter other than the 600 s wall-clock limit was modified: in particular, random seeds were left at their default values so that the parallel multi-worker search of each engine runs

exactly as a practitioner would obtain it out of the box. Each instance was solved *once* by each solver; we did not average over repeated runs because the engines are deterministic for a fixed seed and thread count on a given machine, and because a single out-of-the-box run is the quantity of practical interest. Any residual non-determinism arising from parallel search therefore reflects the genuine behaviour of the default configuration rather than a tuned best-of-k result. Regarding the treatment of terminal states: when a solver reaches the time limit, the best feasible solution and the best dual bound found so far are recorded as its UB and LB, respectively; when a solver fails to find any feasible solution within the budget (which occurs only on the largest 1000×1000 DCT instances for the interval-based engines), the trivial horizon-based value $H = \sum_{j,k} p_{j,k}$ is used as the reported UB, and the absence of a non-trivial dual bound is recorded as a missing LB and excluded from the statistical analysis via listwise deletion (see Section 4.2). To support full verifiability, the complete instance-level results—objective values, bounds, solver statuses, and runtimes for every solver and instance—are made publicly available in a dedicated repository at https://github.com/fjyuraszeck/CP-Benchmark (accessed on 5 February 2026). We stress that this design deliberately measures *out-of-the-box* performance: default parameters and official example formulations are used precisely so that the comparison reflects the experience of a non-expert practitioner, and the resulting ranking should therefore be read as an out-of-the-box ranking rather than as the best attainable performance of each engine after tuning or alternative modelling.

A word is also in order on the choice of a single uniform 600 s budget for all instances irrespective of their size. The aim of our experiments is not to optimise each instance to the fullest—which would call for an instance-dependent and possibly much larger budget—but to compare the solvers under a common deliberately constrained amount of resources. A fixed budget that is generous for small instances yet clearly insufficient for the largest ones lets us probe the solvers as "intuition gainers": it reproduces the realistic situation in which a practitioner facing a large or hard instance has only a limited time to obtain a usable good-quality feasible schedule. We acknowledge that solver rankings may shift under substantially shorter or longer budgets. In particular, longer budgets would tend to favour the metaheuristic engine of Hexaly, whose incumbent keeps improving monotonically over time. We make this scope explicit so that the reported ranking is understood as conditional on a uniform moderate time budget rather than as an asymptotic statement.

For evaluation purposes, we adopt three established performance indicators: the optimality gap, the relative percent deviation with respect to the best-known upper bound, and the relative percent deviation with respect to the best-known lower bound. They are defined, respectively, as

$$\mathrm{GAP} = \frac{\mathrm{UB} - \mathrm{LB}}{\mathrm{UB}}, \quad \mathrm{RPD}_{\mathrm{UB}} = \frac{\mathrm{UB} - \mathrm{Best}_{\mathrm{UB}}}{\mathrm{Best}_{\mathrm{UB}}}, \quad \mathrm{RPD}_{\mathrm{LB}} = \frac{\mathrm{Best}_{\mathrm{LB}} - \mathrm{LB}}{\mathrm{Best}_{\mathrm{LB}}}. \tag{6}$$

Here UB and LB denote the best upper and lower bounds delivered by the solver within the time limit, and $\mathrm{Best}_{\mathrm{UB}}$, $\mathrm{Best}_{\mathrm{LB}}$ denote the best-known values reported in the literature, including the recent improvements obtained from CP-based bounding procedures [43,44] and the public Shylo repository for Taillard. Negative values of $\mathrm{RPD}_{\mathrm{UB}}$ or $\mathrm{RPD}_{\mathrm{LB}}$ indicate that a solver has produced a bound that is strictly better than the best-known one, that is, a new state-of-the-art bound for the corresponding instance.

4. Results

This section reports and discusses the computational experiments. We first describe the testbed and the experimental setting (Section 4.1), then present aggregate results across the 332 instances of the nine benchmark families considered (Section 4.2), and finally provide a

per-family analysis with an emphasis on the largest industrially relevant Da Col–Teppan instances (Section 4.3). The new best-known bounds are highlighted in Section 4.4.

4.1. Testbed and Experimental Setting

All the computational experiments were carried out on a single workstation equipped with an Intel Core i7-10750H processor (six physical cores, 12 threads, and 2.6 GHz base clock) and 16 GB of 2933 MHz DDR4 RAM, running under default operating system conditions. As stated in Section 3.4, every solver runs with a wall-clock time limit of 600 s per instance and uses all available threads. The experimental matrix consists of $4 \times 332 = 1328$ runs, totalling more than 220 h of pure solver time.

The 332 problems are drawn from nine canonical benchmark families: Fisher–Thompson (three instances), Lawrence (40), Adams–Balas–Zawack (five), Applegate–Cook (10), Yamada–Nakano (four), Storer–Wu–Vaccari (20), Taillard (80), Demirkol–Mehta–Uzsoy (80), and Da Col–Teppan (90). They span sizes from 6×5 to 1000×1000, covering more than four orders of magnitude in the number of operations.

4.2. Aggregate Results

Table 2 reports, for each solver, the number of instances solved to proven optimality (#OPT), the average optimality gap, the average relative percent deviations with respect to the best-known upper and lower bounds, and the average solver runtime. The same indicators restricted to the subset of instances solved to optimality are reported in the rightmost columns. Detailed results per instance are available at https://github.com/fjyuraszeck/CP-Benchmark (accessed on 5 February 2026).

Table 2. Aggregate performance of the four CP solvers across the 332 JSSP instances. Best values per column in bold.

Solver	#OPT	Avg. GAP (%)	Std. GAP (%)	Avg. RPD_{UB} (%)	Std. RPD_{UB} (%)	Avg. RPD_{LB} (%)	Std. RPD_{LB} (%)	Avg. Time (s)
CP Optimizer 22.1.2	166	6.85	16.14	1.88	3.34	4.15	15.38	326.7
OR-Tools 9.14.6206	144	7.21	17.91	288.85 [a]	1601.55 [a]	3.59	15.32	364.7
Hexaly 14.0	116	6.33	8.36	2.37	3.30	**1.94**	3.05	405.4
OptalCP 2026.2.0	**191**	**3.55**	**7.99**	**0.76**	**1.37**	0.58 [b]	**1.10**	**272.1**

[a] The high average is driven by 10 outlier 1000×100 Da Col–Teppan instances on which OR-Tools fails to escape the trivial horizon-based primal solution within 600 s. The median RPD_{UB} for OR-Tools is 0.65%. It is worth noting that this value is 0.74% for Hexaly and 0.00% for both CP Optimizer and OptalCP. [b] OptalCP attains the smallest RPD_{LB} overall when the 1000×1000 instances, on which it does not deliver a non-trivial dual bound, are excluded; on the full set its average is artificially favourable because of the same large-instance effect.

OptalCP achieves the largest number of certificates of optimality (191/332, 57.5%), followed by CP Optimizer (166, 50.0%), OR-Tools (144, 43.4%), and Hexaly (116, 34.9%). OptalCP also dominates the aggregate average optimality gap (3.55%) and the average primal deviation RPD_{UB} (0.76%). On instances where the solver does prove optimality, OptalCP is the fastest on average (30.1 s vs. 43.1 s for Hexaly, 53.5 s for CP Optimizer, and 57.4 s for OR-Tools); the median runtime is 2–3 s for all four engines, indicating a heavy-tailed runtime distribution dominated by the hardest instances. Additionally, it can be observed that the dominance of OptalCP over the other solvers is not limited to the average values of the performance indicators but is also reflected in lower standard deviations, highlighting the robustness and consistency of its performance.

A complementary view is given in Table 3, which reports the number of instances on which each solver is the unique strict winner with respect to (i) the optimality gap and

(ii) the upper bound RPD_{UB}. The latter is particularly informative because it captures primal performance, that is, the practical quality of the schedule delivered by the solver.

Table 3. Solver wins by performance indicator. "Strict wins" counts instances on which a single solver achieves the best value; "shared" counts instances on which a solver attains the best value but is tied with at least one other solver. The best values are higlithed in bold letter.

Solver	Strict GAP Wins	Shared GAP Wins	Strict RPD_{UB} Wins	New Best UBs
CP Optimizer 22.1.2	4	168	5	7
OR-Tools 9.14.6206	0	144	4	0
Hexaly 14.0	27	116	27	**22**
OptalCP 2026.2.0	**132**	**169**	**125**	2

OptalCP is the strict winner on the largest share of instances (132 of 332, 39.8%) and is co-optimal on a further 169. Hexaly comes second in strict wins (27), and it is the engine that improves the largest number of best-known upper bounds (22, almost all on the very large 1000×1000 instances). CP Optimizer is the strict winner on four instances and improves seven best-known UBs, while OR-Tools is never the strict winner with respect to GAP and never improves a best-known bound, although it remains co-optimal on 144 instances and is highly competitive on small- and medium-sized problems.

A graphical summary of the distribution of GAP, RPD_{UB}, RPD_{LB}, and runtime across the four solvers is provided in Figure 1, and it is consistent with the aggregated indicators above.

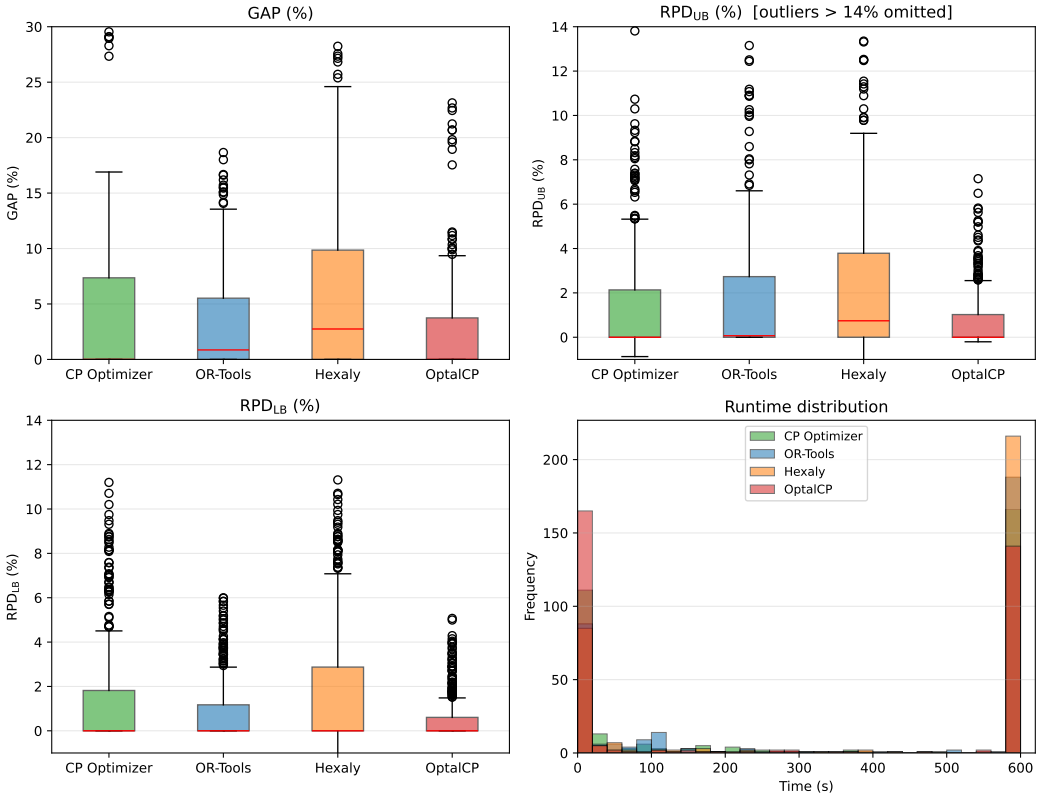


Figure 1. Boxplots of GAP, RPD_{UB}, and RPD_{LB}, and histogram of runtimes for the four CP solvers on the 332 JSSP instances. Outliers are defined according to Tukey's rule ($1.5 \cdot IQR$). The runtime histogram is capped at the 600 s budget per instance.

Additionally, a statistical comparison of optimality gaps across the four solvers was conducted using the Friedman test, a non-parametric procedure that is appropriate for repeated-measures designs in which the same set of instances is evaluated by all the competing methods. Prior to testing, instances with missing observations for any solver were excluded from the analysis using listwise deletion. Specifically, 20 instances were removed, all belonging to the DaCol and Teppan benchmark set with 1000 jobs and either 100 or 1000 machines, as at least one solver failed to report a valid bound for these cases. The analysis was thus conducted on the remaining 312 instances. The Friedman test produced a highly significant result ($\chi^2 = 309.73$, $df = 3$, $p < 0.001$), providing strong evidence that the four solvers do not perform equivalently with respect to optimality gap. To identify which pairs of solvers differ significantly, post hoc comparisons were performed using the Nemenyi test, which controls for family-wise error rate in all pairwise comparisons. At a significance level of $\alpha = 0.05$, the critical difference was $CD = 0.266$. The solvers were ranked by average rank across instances, yielding values of 1.75 for OptalCP, 2.46 for CP Optimizer, 2.67 for OR Tools, and 3.12 for Hexaly, where a lower rank corresponds to a smaller optimality gap. The Nemenyi test indicated that OptalCP achieved significantly lower gaps than all the other solvers ($p < 0.001$ in each case). Although CP Optimizer and OR Tools did not differ significantly from one another ($p = 0.145$), both solvers outperformed Hexaly by a statistically significant margin ($p < 0.001$). Hexaly recorded the highest average optimality gap and ranked last among all the solvers evaluated. Taken together, these findings establish a clear performance hierarchy: OptalCP delivers the best bound quality, followed by CP Optimizer and OR Tools at a comparable level, with Hexaly trailing significantly behind.

4.3. Per-Family Analysis

Table 4 reports the same four indicators broken down by benchmark family. The table reveals strong family-dependent effects.

Table 4. Per-family aggregated results: number of instances solved to optimality (#OPT) and average GAP (%) for each solver. The best value per row is highlighted in bold.

Family	# inst.	CP Optimizer		OR-Tools		Hexaly		OptalCP	
		#OPT	GAP %	#OPT	GAP %	#OPT	GAP %	#OPT	GAP %
Fisher–Thompson	3	**3**	**0.00**	**3**	**0.00**	2	2.58	**3**	**0.00**
Lawrence	40	39	0.07	39	0.07	26	1.49	**40**	**0.00**
Adams–Balas–Zawack	5	2	4.35	2	2.46	1	6.90	**3**	**1.10**
Applegate–Cook	10	**10**	**0.00**	**10**	**0.00**	0	6.73	**10**	**0.00**
Yamada–Nakano	4	0	12.02	0	6.71	0	12.34	**0**	**5.00**
Storer–Wu–Vaccari	20	8	4.30	7	3.60	5	5.72	**11**	**1.73**
Taillard	80	38	3.06	23	2.86	21	4.93	**47**	**1.24**
Demirkol–Mehta–Uzsoy	80	16	7.17	10	7.43	11	8.62	**27**	**4.05**
Da Col–Teppan	90	50	15.35	50	17.32	**50**	**7.60**	50	8.23
Total/Average	**332**	**166**	**6.85**	**144**	**7.21**	**116**	**6.33**	**191**	**3.55**

Several observations are worth noting. First, on the small classical instances (Fisher–Thompson, Applegate–Cook, and Lawrence) all the solvers are very strong, but OptalCP is the only engine that closes *all* 40 Lawrence instances and matches OR-Tools and CP Optimizer on Fisher–Thompson and Applegate–Cook. Hexaly, by contrast, struggles on these small problems because its hybrid local-search engine is not specialised for small but structurally tight instances. Second, on the medium-difficulty Storer–Wu–Vaccari and Taillard sets, OptalCP clearly dominates: it certifies 11 SWV optima (vs. eight for CP Optimizer) and 47 Taillard optima (vs. 38 for CP Optimizer), and it reduces the average GAP

on Taillard to only 1.24%. Third, the hardest medium-size family in our experiments is Demirkol–Mehta–Uzsoy: every solver leaves a substantial residual gap, but OptalCP again leads (27 optima, 4.05% GAP) by a wide margin.

The Yamada–Nakano 20×20 instances are the only family on which no solver proves optimality within 600 s. OptalCP attains the smallest average GAP (5.00%), followed by OR-Tools (6.71%); CP Optimizer and Hexaly trail at $\sim$12% and present near-identical residual gaps, suggesting that, on tight 20×20 structures, the lower-bounding mechanisms of CP Optimizer and Hexaly converge slowly.

The most striking results occur on the Da Col–Teppan family, which contains the only truly industrial-size instances of the testbed (up to 1000×1000). Here Hexaly and OptalCP report the smallest deviations from the best-known upper bounds (RPD_{UB} of 0.80% and 0.54%, respectively), and they are also the only two solvers that always return a feasible solution: on the 10 instances of size 1000×1000, neither the CP Optimizer nor OR-Tools is able to produce a non-trivial primal solution within the time budget, while Hexaly and OptalCP both deliver schedules within a few percent of the literature reference. CP Optimizer and OR-Tools nevertheless certify optimality on the 50 small to medium DCT instances on which all the solvers happen to agree on the objective value, so their #OPT count is identical (50) to that of Hexaly and OptalCP. The disparities on this family expose, in a controlled manner, the practical impact of the architectural differences described in Section 3.3: classical CP solvers based on interval-variable propagation require numbers of variables and constraints that grow quadratically with $n \cdot m$, whereas list-based and metaheuristic engines such as Hexaly scale much more gracefully.

4.4. New Best-Known Bounds

A direct and tangible by-product of running four state-of-the-art CP solvers on a uniform testbed is the identification of new best-known bounds, which directly contribute to the literature on the JSSP. In total, our experiments yield new best-known upper bounds (NBKUBs) on **22 distinct instances** and **one new best-known lower bound** (NBKLB) across the 332 instances. These 22 instance-level improvements correspond to 31 solver-level improvement events since, in several instances, more than one solver independently beats the previous best-known upper bound (cf. Table 5). By solver, the breakdown of improvement events is as follows: Hexaly produces 22 NBKUBs and the only NBKLB; CP Optimizer produces seven NBKUBs; OptalCP produces two NBKUBs; and OR-Tools does not improve any best-known bound within the time budget. All the improvements concentrate on the Da Col–Teppan family, which is consistent with the fact that DCT instances are the most recent and least-explored set in our testbed. The detailed list of new bounds is reported in Table 5.

Before continuing the analysis, we would like to note that, as depicted in Table 5, for some instances, more than one solver improves upon the best-known upper bound. As a result, the total number of improvements reported for individual solvers does not correspond to the number of distinct instances for which improvements were achieved. Counting each instance only once, new best-known upper bounds are obtained for 22 instances.

Additionally, the geographical concentration of these improvements on the Da Col–Teppan family deserves comment. The DCT generator was introduced relatively recently [19,27], and the corresponding instances have not yet been the object of as much algorithmic refinement as the Taillard or Demirkol–Mehta–Uzsoy sets, which leaves more space for primal improvement. Hexaly, with its list-based metaheuristic engine specifically engineered for very large search spaces, is unsurprisingly the engine that benefits the most. Two architectural traits are particularly helpful in this regime: (i) the ability to produce a feasible solution within seconds even for 1000×1000 instances, which classical interval-

based CP solvers cannot match within 600 s; and (ii) the ability to keep improving that solution by means of large-neighbourhood moves on list variables, which scale linearly with the number of operations.

Table 5. New best-known bounds (UB and LB) obtained in this work. "Old" denotes the previously best-known value reported in the literature; "new" denotes the value obtained by the indicated solver within 600 s; Δ % is the relative improvement, computed as $(\text{Old} - \text{New})/\text{Old} \cdot 100$ for UBs and $(\text{New} - \text{Old})/\text{Old} \cdot 100$ for LBs. All instances belong to the Da Col–Teppan family.

Instance	n	m	Bound	Solver	Old	New	Δ %
New best-known upper bound (UB)							
tai_j100_m1000_1	100	1000	UB	CP Optimizer	544,732	544,381	0.064
tai_j100_m1000_1	100	1000	UB	Hexaly	544,732	542,479	0.414
tai_j100_m1000_2	100	1000	UB	CP Optimizer	546,598	546,359	0.044
tai_j100_m1000_2	100	1000	UB	Hexaly	546,598	545,155	0.264
tai_j100_m1000_3	100	1000	UB	CP Optimizer	549,372	545,312	0.739
tai_j100_m1000_3	100	1000	UB	Hexaly	549,372	547,367	0.365
tai_j100_m1000_4	100	1000	UB	Hexaly	545,138	542,985	0.395
tai_j100_m1000_5	100	1000	UB	Hexaly	545,535	542,170	0.617
tai_j100_m1000_5	100	1000	UB	OptalCP	545,535	544,417	0.205
tai_j100_m1000_6	100	1000	UB	CP Optimizer	545,730	544,013	0.315
tai_j100_m1000_6	100	1000	UB	Hexaly	545,730	542,938	0.512
tai_j100_m1000_7	100	1000	UB	CP Optimizer	546,899	546,383	0.094
tai_j100_m1000_7	100	1000	UB	Hexaly	546,899	541,887	0.917
tai_j100_m1000_8	100	1000	UB	CP Optimizer	549,337	545,466	0.705
tai_j100_m1000_8	100	1000	UB	Hexaly	549,337	545,671	0.667
tai_j100_m1000_8	100	1000	UB	OptalCP	549,337	548,419	0.167
tai_j100_m1000_9	100	1000	UB	CP Optimizer	550,693	545,891	0.872
tai_j100_m1000_9	100	1000	UB	Hexaly	550,693	546,749	0.716
tai_j100_m1000_10	100	1000	UB	Hexaly	543,797	541,528	0.417
tai_j1000_m100_1	1000	100	UB	Hexaly	539,120	526,261	2.385
tai_j1000_m100_2	1000	100	UB	Hexaly	540,895	532,860	1.486
tai_j1000_m100_3	1000	100	UB	Hexaly	534,794	529,027	1.078
tai_j1000_m100_4	1000	100	UB	Hexaly	536,317	531,407	0.916
tai_j1000_m100_5	1000	100	UB	Hexaly	532,016	531,581	0.082
tai_j1000_m100_6	1000	100	UB	Hexaly	535,189	525,729	1.768
tai_j1000_m100_7	1000	100	UB	Hexaly	535,894	531,623	0.797
tai_j1000_m100_8	1000	100	UB	Hexaly	533,985	527,056	1.297
tai_j1000_m100_9	1000	100	UB	Hexaly	539,511	533,251	1.160
tai_j1000_m100_10	1000	100	UB	Hexaly	540,884	533,151	1.430
tai_j1000_m1000_1	1000	1000	UB	Hexaly	877,062	877,052	0.001
tai_j1000_m1000_3	1000	1000	UB	Hexaly	878,805	878,296	0.058
New best-known lower bound (LB)							
tai_j1000_m1000_10	1000	1000	LB	Hexaly	541,530	549,075	1.394

CP Optimizer's seven UB improvements all occur on 100×1000 instances, where the number of jobs is small relative to the number of machines and the structure of the schedule is closer to that of a parallel machine problem. In this regime, CP Optimizer's failure-directed search and large-neighbourhood mechanisms are still able to escape weak primal solutions and improve them, although it cannot match Hexaly when the number of jobs grows to 1000. OptalCP's two UB improvements, both on 100×1000 instances, support a similar reading: the OptalCP architecture inherits from CP Optimizer the strong propagation engine but adds a more aggressive parallel-search portfolio with reinforcement-learning acceleration, which appears to translate into competitive primal performance even on instances where the number of operations is substantial.

Finally, on the dual side, the single new LB improvement obtained by Hexaly on `tai_j1000_m1000_10` (a 1.4% increase in the dual bound) is a noteworthy outlier: dual

bounds for industrial-size JSSPs are very difficult to strengthen and are typically the domain of specialised CP-based bounding procedures rather than off-the-shelf solvers [43,44]. We attribute this improvement to the combination of Hexaly's local consistency checks with its rapid feasibility detection, which together can occasionally close primal–dual gaps that elude pure interval-propagation engines.

Overall, the empirical evidence supports three preliminary conclusions: (i) OptalCP currently offers the best primal–dual trade-off across heterogeneous JSSP instances; (ii) Hexaly remains the engine of choice for industrial-scale problems where producing *any* high-quality feasible solution is the binding constraint; and (iii) CP Optimizer and OR-Tools, while consistently strong on classical benchmark families, are increasingly challenged by the new generation of CP solvers represented by OptalCP and Hexaly.

4.5. Performance vs. Instance Size and the n/m Ratio

The per-family analysis of Section 4.3 mixes effects that are due to the structural properties of each benchmark with effects that are due to the size of its instances. To disentangle the two, in this subsection, we analyse solver performance along two complementary axes: the size of the instance, measured by the total number of operations $n \cdot m$, and the ratio between the number of jobs and the number of machines, n/m.

4.5.1. Effect of Instance Size

Table 6 reports, for six size buckets ranging from *tiny* (less than 100 operations) to *industrial* (more than 50,000 operations), the number of optimality certificates and the average optimality gap delivered by each solver. The buckets are designed to uniformly span the range of operation counts present in our testbed.

Table 6. Performance by instance size, measured by total number of operations $n \cdot m$. "#OPT" is the number of instances solved to proven optimality; "GAP %" is the average optimality gap. Best values per row are highlighted in bold. "Industrial" contains the 100×1000, 1000×100, and 1000×1000 DCT instances (30 instances).

Size Bucket ($n \cdot m$)	# inst.	CP Optimizer		OR-Tools		Hexaly		OptalCP	
		#OPT	GAP %	#OPT	GAP %	#OPT	GAP %	#OPT	GAP %
Tiny (<100)	45	**45**	**0.00**	**45**	**0.00**	31	2.17	**45**	**0.00**
Small (100–500)	117	40	5.12	40	3.52	13	7.23	**60**	**1.96**
Medium (500–1500)	100	51	3.78	39	4.95	42	4.73	**56**	**2.54**
Large (1500–5000)	10	**10**	**0.00**	0	3.06	**10**	**0.00**	**10**	**0.00**
Very Large (5000–50,000)	30	20	9.97	20	10.88	20	8.70	**20**	**6.89**
Industrial (>50,000)	30	0	46.45	0	52.94	0	**14.09**	0	22.60
Total/Avg.	**332**	**166**	**6.85**	**144**	**7.21**	**116**	**6.33**	**191**	**3.55**

The table reveals a clear pattern. On *tiny* instances ($n \cdot m < 100$), the three interval-based CP solvers (CP Optimizer, OR-Tools, and OptalCP) close all 45 problems within the time budget, while Hexaly closes only 31. This confirms that pure CP propagation is overwhelmingly competitive on small but tightly constrained instances, where the search-tree size is the dominant determinant of runtime. The picture changes dramatically in the *small* (100–500 operations) bucket, which contains 117 problems mostly drawn from the Demirkol–Mehta–Uzsoy and Storer–Wu–Vaccari families: OptalCP closes 60 of them with an average GAP of 1.96%, while the CP Optimizer and OR-Tools close only 40 each. As the size grows further, OptalCP retains the lead on *medium* (500–1500) and *very large* (5 K–50 K) instances, but CP Optimizer narrows the gap and Hexaly emerges as a strong alternative.

The most striking transition occurs in the *industrial* bucket ($n \cdot m > 50{,}000$ operations, mostly the 1000×100 and 1000×1000 DCT instances). Here, none of the four solvers proves

optimality, but Hexaly delivers an average GAP of 14.09%, compared to 22.60% for OptalCP, 46.45% for CP Optimizer, and 52.94% for OR-Tools. This is the most relevant region of the parameter space for industrial deployment, and it confirms that pure interval-based CP propagation does not scale gracefully to instances with hundreds of thousands of operations regardless of how strong the search engine is. Conversely, the metaheuristic-driven engine of Hexaly clearly retains its primal effectiveness in this regime.

4.5.2. Effect of the n/m Ratio and the Square ($n = m$) Case

A second important structural property of a JSSP instance is the ratio between the number of jobs and the number of machines, n/m. Loosely, n/m measures how "rectangular" the instance is: low values (e.g., $n/m \approx 0.1$) correspond to instances with many machines and few jobs, intermediate values ($n/m \approx 1$) correspond to *square* instances in which the disjunctive graph is nearly balanced, and high values ($n/m \gg 1$) correspond to instances with many jobs flowing through a small number of machines, which approximates a flow-shop-like structure.

The JSSP literature has long observed that square instances ($n \approx m$) are typically the hardest [9,39,51]: when the number of jobs equals the number of machines, the disjunctive graph is densest and the symmetries between rows and columns of the schedule maximise the search-tree size. Our testbed contains 88 instances with $0.9 \leq n/m \leq 1.1$ (including 88 strictly square problems with $n = m$), allowing us to test this conjecture systematically across all four solvers.

Three findings emerge from Table 7.

Table 7. Performance by the n/m ratio. "Square" instances ($0.9 \leq n/m \leq 1.1$) are isolated in their own bucket; instances with $n/m > 5$ approximate a flow-shop structure and are dominated by the 1000×100 DCT problems. Best values per row are in bold.

Ratio n/m	# inst.	CP Optimizer		OR-Tools		Hexaly		OptalCP	
		#OPT	GAP %	#OPT	GAP %	#OPT	GAP %	#OPT	GAP %
$n/m < 0.5$ (machine-heavy)	30	20	1.26	20	2.00	**20**	**1.16**	20	1.40
$n/m \in [0.9, 1.1]$ (square)	88	43	7.30	43	6.22	15	12.95	**49**	**8.19** [a]
$1.1 < n/m \leq 2.5$	118	31	5.09	21	5.15	11	6.69	**49**	**2.78**
$2.5 < n/m \leq 5$	66	52	1.60	40	2.58	50	1.91	**53**	**0.77**
$n/m > 5$ (job-heavy)	30	20	29.71	20	33.29	20	**0.34**	20	**0.00**
Total/Avg.	**332**	**166**	**6.85**	**144**	**7.21**	**116**	**6.33**	**191**	**3.55**

[a] For square instances, OR-Tools attains a marginally smaller average GAP (6.22%) than OptalCP (8.19%), but OptalCP still produces 6 more proven optima (49 vs. 43). The slightly worse residual GAP is concentrated on the very hard Yamada–Nakano 20×20 instances.

Square instances are the hardest ones. The 88 square instances ($n/m \approx 1$) have the largest residual gap on average ($\sim$ 6–13% depending on the solver) and the smallest ratio of optimality certificates per instance ($49/88 = 55.7\%$ for the best solver, OptalCP). This confirms the long-standing observation that the square case is the hardest one in the $J \,||\, C_{\max}$ literature. The intuition is well-known [39]: when $n = m$, every machine must process exactly n operations, every job must visit exactly $m = n$ machines, the load is perfectly balanced, and the makespan-minimisation problem cannot be "decomposed" along either the job or the machine dimension. As a consequence, the search must consider the full set of $n!$ orderings on each machine, with little space for early pruning. In our experiments, this regime is the most discriminating between the three interval-based CP solvers (CP Optimizer, OR-Tools, and OptalCP), which all obtain similar numbers of optima (43–49), and Hexaly (only 15 optima), whose metaheuristic engine appears to be less effective when the disjunctive structure is balanced.

Rectangular instances become progressively easier. Moving away from $n/m = 1$, the picture improves rapidly. Already in the $1.1 < n/m \leq 2.5$ regime, OptalCP closes 49 of 118 instances and brings the average GAP down to 2.78%, with CP Optimizer as the second-best engine at 5.09%. In the $2.5 < n/m \leq 5$ regime, dominated by the Lawrence and Adams–Balas–Zawack instances, all four solvers solve more than 60% of the problems and the residual GAP falls below 3%. The interval-based propagation routines benefit from the asymmetry because the smaller dimension (the number of machines) imposes tight constraints that prune the search tree.

Extreme ratios reveal strong specialisation effects. For $n/m < 0.5$ (i.e., very few jobs and many machines, dominated by the 100×1000 DCT instances) all the solvers tie on #OPT, but the average GAP varies from 1.16% (Hexaly) to 2.00% (OR-Tools), with Hexaly leading because its local-search engine handles the parallel-machine-like structure with ease. The $n/m > 5$ regime, dominated by the 1000×100 DCT instances, is the most polarised case in our testbed: Hexaly and OptalCP achieve average GAPs of 0.34% and 0.00%, respectively, while CP Optimizer and OR-Tools both exceed 30%. Hexaly produces here the bulk of its 22 NBKUBs. Architecturally, when $n \gg m$, the schedule starts to resemble a permutation flow-shop instance, and metaheuristic engines—which can exchange entire job sub-sequences in a single move—are dramatically more effective than interval-propagation-based search.

Taken together, the size-based and ratio-based analyses point to a single take-away message: the relative competitiveness of CP solvers on the JSSP depends on the size and structural "shape" of the instance much more than on the solver brand alone. OptalCP appears to be the most robust engine across all regimes, but each of the other three solvers retains a regime in which it is the most appropriate choice: CP Optimizer and OR-Tools for tiny tightly constrained problems, OptalCP for medium to large problems with rectangular structure, and Hexaly for industrial-size problems with extreme aspect ratios.

5. Discussion

The empirical evidence collected in Section 4 allows us to revisit, with more nuance, the broader research questions that motivated this work. We organise the discussion around five themes: solver competitiveness, the role of problem size and structure, the contribution to the JSSP literature, methodological lessons for benchmarking CP solvers, and the limitations of our study.

Solver competitiveness across regimes. Our experiments show that no single solver is uniformly dominant across the 332 instances of the testbed; rather, each engine has a regime in which it is the most appropriate choice. OptalCP [24] emerges as the most robust solver overall, achieving the largest number of proven optima (191/332, 57.5%), the smallest average optimality gap (3.55%), and the smallest average runtime when proving optimality (30.1 s). This is consistent with two architectural features that distinguish OptalCP from earlier CP engines: a parallel-search portfolio that combines multiple variants of failure-directed search [16], large-neighbourhood search, and asynchronously supplied user heuristics; and a reinforcement-learning-based controller that adaptively allocates a search effort across the portfolio [17]. CP Optimizer [15], which shares the FDS lineage with OptalCP, remains very competitive on small- and medium-sized instances but is increasingly outperformed by OptalCP on harder problems. OR-Tools (CP-SAT) [21] is highly competitive on tiny and small instances, where its SAT-based reasoning is most effective, but loses ground on medium to large problems. Hexaly [22,23] is dominated on small- and medium-sized instances but becomes the strongest engine on the truly industrial 1000×100 and 1000×1000 DCT problems, where it produces the bulk of the 22 new best-known upper bounds.

These results refine, rather than contradict, the conclusions of the previous conference paper [26]. On the Taillard set alone, that earlier study identified CP Optimizer as the strongest engine. The present work confirms CP Optimizer's strength on Taillard but shows that this conclusion does not generalise to other instance families and to the larger sizes considered here. The introduction of OptalCP into the comparison further reshapes the ranking: OptalCP closes 47 Taillard instances (vs. 38 for CP Optimizer) and dominates seven of the nine families considered. Practitioners interested in CP-based scheduling should therefore be aware that the most appropriate solver for a given application depends not only on the family of the problem but also on its size and the n/m ratio.

Problem size and structure. The size-based analysis (Section 4.5.1) reveals that pure interval-propagation-based CP engines do not scale gracefully beyond a few thousand operations. CP Optimizer, OR-Tools, and OptalCP all fail to produce a non-trivial primal solution on the ten 1000×1000 DCT instances within 600 s, while Hexaly returns feasible solutions within seconds in every case. The n/m ratio analysis (Section 4.5.2) further shows that square instances are the hardest, in line with the long-standing folklore in the metaheuristic literature [9,39], while strongly rectangular instances (either $n/m \ll 1$ or $n/m \gg 1$) are easier and benefit different solvers in different ways. The combination of these two analyses suggests a simple heuristic rule for solver selection in industrial applications: use OptalCP when the schedule has up to a few thousand operations and an aspect ratio not too far from one and use Hexaly when the schedule has tens or hundreds of thousands of operations or when the aspect ratio is extreme. Our results align with the recent algorithm-selection literature for CP scheduling [25], which advocates choosing the solver based on instance features rather than treating any single engine as the default.

Contribution to the JSSP literature. Beyond the methodological findings, this work contributes new best-known upper bounds on 22 distinct instances (corresponding to 31 solver-level improvement events) and one new best-known lower bound, all on the Da Col–Teppan family. The Da Col–Teppan instances are the most recent industrial-size testbed for the JSSP [19,27], and the fact that they are still amenable to substantial primal improvement after a single 600 s run with off-the-shelf CP technology highlights both their value as a benchmark and the rapid progress of the CP-for-scheduling community. The single new lower bound on `tai_j1000_m1000_10`, although small in relative terms, is a noteworthy result given that dual-bound improvements on industrial-scale JSSPs are typically the domain of specialised CP-based bounding procedures [43,44]. Together, these new bounds extend the accuracy of the literature reference values and will support future comparative studies.

Methodological lessons. Several methodological lessons emerge from our experimental design. First, evaluating solvers on a single instance family produces partial and potentially misleading conclusions. The introduction of nine families spanning four orders of magnitude in size has been essential to expose the qualitative differences between engines that all looked competitive on Taillard. Second, the choice of indicators matters: the average optimality gap and the count of proven optima provide a coarse first impression, but a finer reading requires considering the deviation from the best-known UBs and LBs, the runtime distribution, and the number of strict wins. Third, performance varies sharply with structural parameters such as size and n/m ratio, which suggests that future benchmarking studies should systematically report results stratified by these parameters rather than only by family. Finally, controlled hardware and identical default parameters are necessary conditions for a fair empirical comparison, and we have made every effort to satisfy them in our setup.

Limitations. Our study also has some limitations that should be acknowledged. All the experiments were run on a single mid-range workstation with six physical cores; since

CP solvers exploit parallelism differently, absolute runtimes—and possibly the relative ranking on the hardest instances—could differ on high-core-count industrial servers. The findings and conclusions reported here should therefore be understood as conditional on the specific experimental design and hardware described in Section 4.1. We restrict the analysis to the canonical $J \,||\, C_{\max}$ formulation; richer scheduling problems—including the flexible job shop [10,25], problems with sequence-dependent setup times, multi-objective formulations, or stochastic variants—may yield different solver rankings. We use default solver parameters in all the experiments; non-trivial tuning, especially for OR-Tools (which exposes a large set of CP-SAT parameters) and Hexaly (which can be extensively configured for specific scheduling tasks), might alter the relative ranking. We do not consider hybrid solver portfolios, in which an off-the-shelf orchestrator runs more than one engine in parallel and returns the best solution found across them; preliminary evidence suggests that such portfolios can be highly effective [19,25]. Finally, our 600 s time budget is shorter than what is sometimes used in the metaheuristic literature; longer budgets would likely benefit Hexaly the most since its metaheuristic engine continues to improve solutions monotonically.

Practical recommendations. Distilling the empirical evidence into actionable guidance, we suggest the following decision rule for practitioners who must pick a single CP engine under a tight time budget. For instances up to a few thousand operations whose aspect ratio is not too far from one—the regime that covers most classical academic benchmarks and many medium-sized industrial cells—OptalCP is the recommended default as it proves optimality most often and yields the smallest gaps. For very small tightly constrained instances, CP Optimizer and OR-Tools are equally safe choices and are freely or openly available. For industrial-scale instances with tens of thousands of operations or with extreme n/m ratios (very few jobs on many machines or vice versa), Hexaly is the recommended engine because it is the only one that reliably returns a high-quality feasible schedule within seconds. When the deployment budget allows running two engines in parallel, the OptalCP–Hexaly pair is the most complementary portfolio and dominates either engine alone. We also stress that these recommendations are derived for the makespan objective; many industrial settings instead optimise total weighted tardiness, total flow time, energy consumption, or several of these simultaneously. Because solver rankings need not transfer across objectives, the present recommendations should be regarded as a well-supported starting point for makespan-driven applications and as a methodological template—rather than a final verdict—for the broader family of scheduling objectives, whose systematic evaluation we leave to future work.

6. Conclusions

This article has presented the largest comparative empirical evaluation, to the best of our knowledge, of state-of-the-art CP solvers on the makespan-minimisation job shop scheduling problem. We have benchmarked four engines—IBM ILOG CP Optimizer, Google OR-Tools (CP-SAT), Hexaly, and OptalCP—on 332 instances drawn from nine canonical benchmark families (Fisher–Thompson, Lawrence, Adams–Balas–Zawack, Applegate–Cook, Yamada–Nakano, Storer–Wu–Vaccari, Taillard, Demirkol–Mehta–Uzsoy, and Da Col–Teppan), spanning sizes from 6×6 to 1000×1000 and including the largest industrial-size JSSPs reported in the literature. All the experiments were carried out on a single workstation with default solver parameters and a uniform 600 s wall-clock time budget per instance, ensuring that the resulting comparison is fair and reproducible.

Our experimental campaign yields five main findings. First, OptalCP is the most competitive solver overall, certifying optimality on $191/332 = 57.5\%$ of the instances, with the smallest average optimality gap (3.55%) and the smallest average runtime when proving

optimality. Second, the four engines exhibit strongly regime-dependent performance: CP Optimizer and OR-Tools dominate on tiny instances; OptalCP dominates on small, medium, and large instances; and Hexaly is the most effective on industrial-size problems with extreme n/m ratios. Third, square instances ($n = m$) remain the most difficult for every solver, consistent with the long-standing observation in the metaheuristic literature; Hexaly is the engine most affected by this hardness. Fourth, the experiments yield new best-known upper bounds on 22 distinct instances (31 solver-level improvement events) and one new best-known lower bound, all of them on Da Col–Teppan instances, with Hexaly being the dominant contributor (22 UBs and the only LB). Fifth, the heterogeneity of solver behaviour across families, sizes, and aspect ratios suggests that practitioners should select their CP engine based on the structural features of the problem at hand rather than treating any single solver as a universal default.

These findings open several avenues for future research. The first is the systematic evaluation of CP solver portfolios, in which two or more engines are run in parallel and the best solution found across them is returned to the user; the complementarity between OptalCP (strongest on small/medium structured instances) and Hexaly (strongest on industrial-size problems) makes them natural candidates for such a portfolio. A second avenue concerns the integration of dedicated CP-based lower-bounding procedures [43,44] into the benchmarking workflow, with the goal of certifying optimality on instances where the best-known LB is only marginally weaker than the best-known UB. A third direction is the extension of the empirical methodology to richer scheduling problems, including the flexible job shop [10], problems with sequence-dependent setup times, multi-objective formulations, and energy-aware variants. A fourth direction is the use of algorithm-selection and machine-learning techniques [17,25] to predict, for any given instance, which CP engine is most likely to perform best, possibly building upon the size and n/m-ratio features identified in this work. Finally, a longer-term research line is the design of hybrid CP–metaheuristic engines that combine the strong propagation of OptalCP and CP Optimizer with the scaling properties of Hexaly in order to bridge the performance gap on industrial-scale JSSPs.

Author Contributions: Conceptualisation, D.R.; methodology, F.Y.; software, F.Y.; validation, F.Y., F.W. and D.R.; formal analysis, F.Y. and D.R.; investigation, F.Y., F.W. and D.R.; resources, F.Y. and D.R.; data curation, F.W.; writing—original draft, F.W. and D.R.; writing—review and editing, F.Y., F.W. and D.R.; visualisation, F.Y., F.W. and D.R.; supervision, F.W. and D.R.; project administration, D.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding authors.

Acknowledgments: The authors thank the developers of CP Optimizer, OR-Tools, Hexaly, and OptalCP for making their solvers available for academic evaluation.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CP	Constraint Programming
FDS	Failure-Directed Search
JSSP	Job Shop Scheduling Problem
LB	Lower Bound
LNS	Large-Neighbourhood Search

MILP	Mixed-Integer Linear Programming
NBKUB	New Best-Known Upper Bound
NBKLB	New Best-Known Lower Bound
RPD	Relative Percent Deviation
SAT	Boolean Satisfiability
UB	Upper Bound

References

1. Pinedo, M.L. *Scheduling: Theory, Algorithms, and Systems*, 6th ed.; Springer: Cham, Switzerland, 2022.
2. Framinan, J.M.; Leisten, R.; García, R.R. *Manufacturing Scheduling Systems: An Integrated View on Models, Methods and Tools*; Springer: London, UK, 2014.
3. Zhang, C.; Wu, Y.; Ma, Y.; Song, W.; Le, Z.; Cao, Z.; Zhang, J. A Review on Learning to Solve Combinatorial Optimisation Problems in Manufacturing. *IET Collab. Intell. Manuf.* **2023**, *5*, e12072. [CrossRef]
4. Parente, M.; Figueira, G.; Amorim, P.; Marques, A. Production Scheduling in the Context of Industry 4.0: Review and Trends. *Int. J. Prod. Res.* **2020**, *58*, 5401–5431. [CrossRef]
5. Rossit, D.A.; Tohmé, F.; Frutos, M. Industry 4.0: Smart Scheduling. *Int. J. Prod. Res.* **2019**, *57*, 3802–3813. [CrossRef]
6. Błażewicz, J.; Ecker, K.H.; Pesch, E.; Schmidt, G.; Węglarz, J. *Scheduling Computer and Manufacturing Processes*; Springer: Berlin/Heidelberg, Germany, 1996.
7. Bożek, A.; Werner, F. Flexible job shop scheduling with lot streaming and sublot size optimisation. *Int. J. Prod. Res.* **2018**, *56*, 6391–6411.
8. Garey, M.R.; Johnson, D.S.; Sethi, R. The Complexity of Flowshop and Jobshop Scheduling. *Math. Oper. Res.* **1976**, *1*, 117–129. [CrossRef]
9. Xie, J.; Li, X.; Gao, L.; Gui, L. A Hybrid Algorithm with a New Neighborhood Structure for Job Shop Scheduling Problems. *Comput. Ind. Eng.* **2022**, *169*, 108205. [CrossRef]
10. Dauzère-Pérès, S.; Ding, J.; Shen, L.; Tamssaouet, K. The Flexible Job Shop Scheduling Problem: A Review. *Eur. J. Oper. Res.* **2024**, *314*, 409–432. [CrossRef]
11. Lange, J.; Werner, F. On neighborhood structures and repair techniques for blocking job shop scheduling problems. *Algorithms* **2019**, *12*, 242. [CrossRef]
12. Prata, B.A.; Abreu, L.R.; Nagano, M.S. Applications of Constraint Programming in Production Scheduling Problems: A Descriptive Bibliometric Analysis. *Results Control Optim.* **2024**, *14*, 100350. [CrossRef]
13. Yuraszeck, F.; Mejia, G.; Lüer-Villagra, A. An Adapted Constraint-Programming Formulation of the Resource-Constrained Project Scheduling Problem Applied to the Identical Parallel Machines Group Shop and Mixed Shop Scheduling Problems. *Int. Trans. Oper. Res.* **2025**, *32*, 1422–1441. [CrossRef]
14. Baptiste, P.; Le Pape, C.; Nuijten, W. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*; International Series in Operations Research & Management Science; Springer: Boston, MA, USA, 2001; Volume 39.
15. Laborie, P.; Rogerie, J.; Shaw, P.; Vilím, P. IBM ILOG CP Optimizer for Scheduling: 20+ Years of Scheduling with Constraints at IBM/ILOG. *Constraints* **2018**, *23*, 210–250. [CrossRef]
16. Vilím, P.; Laborie, P.; Shaw, P. Failure-Directed Search for Constraint-Based Scheduling. In *Proceedings of the Integration of AI and OR Techniques in Constraint Programming—CPAIOR 2015*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9075, pp. 437–453. [CrossRef]
17. Heinz, V.; Vilím, P.; Hanzálek, Z. Reinforcement Learning for Search Tree Size Minimization in Constraint Programming: New Results on Scheduling Benchmarks. *Comput. Ind. Eng.* **2025**, *200*, 111413. [CrossRef]
18. Naderi, B.; Ruiz, R.; Roshanaei, V. Mixed-Integer Programming vs. Constraint Programming for Shop Scheduling Problems: New Results and Outlook. *Inf. J. Comput.* **2023**, *35*, 817–843. [CrossRef]
19. Da Col, G.; Teppan, E.C. Industrial-Size Job Shop Scheduling with Constraint Programming. *Oper. Res. Perspect.* **2022**, *9*, 100249. [CrossRef]
20. Google. OR-Tools: Google Optimization Tools. 2025. Available online: https://developers.google.com/optimization (accessed on 1 May 2026).
21. Perron, L.; Didier, F.; Gay, S. The CP-SAT-LP Solver. In *Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*; *LIPIcs*; Schloss Dagstuhl—Leibniz-Zentrum für Informatik: Wadern, Germany, 2023; Volume 280, pp. 3:1–3:2. [CrossRef]
22. Benoist, T.; Estellon, B.; Gardi, F.; Megel, R.; Nouioua, K. LocalSolver 1.x: A Black-Box Local-Search Solver for 0-1 Programming. *4OR* **2011**, *9*, 299–316. [CrossRef]

23. Blaise, L. Modeling Scheduling Problems with Hexaly. In Proceedings of the 11th IFAC Conference on Manufacturing Modelling, Management and Control—IFAC MIM 2025, Trondheim, Norway, 30 June–3 July 2025.
24. ScheduleOpt. *OptalCP: A Modern Constraint Programming Solver for Scheduling*, version 2026.2.0; ScheduleOpt: Melbourne, VIC, Australia, 2026. Available online: https://optalcp.com (accessed on 6 May 2026).
25. Müller, D.; Müller, M.G.; Kress, D.; Pesch, E. An Algorithm Selection Approach for the Flexible Job Shop Scheduling Problem: Choosing Constraint Programming Solvers through Machine Learning. *Eur. J. Oper. Res.* **2022**, *302*, 874–891. [CrossRef]
26. Yuraszeck, F.; Rossit, D.A.; Milović, M.; Córdova, A.; Olivares, G. Benchmarking Constraint Programming Software: Insights from the Job Shop Scheduling Problem. In Proceedings of the 24th LACCEI International Multi-Conference for Engineering, Education, and Technology, Santiago, Chile, 15–17 July 2026.
27. Da Col, G.; Teppan, E.C. Industrial Size Job Shop Scheduling Tackled by Present-Day CP Solvers. In *Proceedings of the Principles and Practice of Constraint Programming—CP 2019*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11802, pp. 144–160. [CrossRef]
28. Beck, J.C.; Feng, T.K.; Watson, J.P. Combining Constraint Programming and Local Search for Job-Shop Scheduling. *Inf. J. Comput.* **2011**, *23*, 1–14. [CrossRef]
29. Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Rinnooy Kan, A.H.G. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Ann. Discret. Math.* **1979**, *5*, 287–326. [CrossRef]
30. Lenstra, J.K.; Rinnooy Kan, A.H.G.; Brucker, P. Complexity of Machine Scheduling Problems. *Ann. Discret. Math.* **1977**, *1*, 343–362. [CrossRef]
31. Manne, A.S. On the Job-Shop Scheduling Problem. *Oper. Res.* **1960**, *8*, 219–223. [CrossRef]
32. Wagner, H.M. An Integer Linear-Programming Model for Machine Scheduling. *Nav. Res. Logist. Q.* **1959**, *6*, 131–140. [CrossRef]
33. Carlier, J.; Pinson, É. An Algorithm for Solving the Job-Shop Problem. *Manag. Sci.* **1989**, *35*, 164–176. [CrossRef]
34. Brucker, P.; Jurisch, B.; Sievers, B. A Branch and Bound Algorithm for the Job-Shop Scheduling Problem. *Discret. Appl. Math.* **1994**, *49*, 107–127. [CrossRef]
35. Martin, P.; Shmoys, D.B. A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem. *Lect. Notes Comput. Sci.* **1996**, *1084*, 389–403. [CrossRef] [PubMed]
36. Van Hoorn, J.J. Dynamic Programming for Routing and Scheduling: Optimizing Sequences of Decisions. Ph.D. Thesis, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands, 2016.
37. Koshimura, M.; Nabeshima, H.; Fujita, H.; Hasegawa, R. Solving Open Job-Shop Scheduling Problems by SAT Encoding. *IEICE Trans. Inf. Syst.* **2010**, *E93-D*, 2316–2318. [CrossRef]
38. Siala, M.; Artigues, C.; Hebrard, E. Two Clause Learning Approaches for Disjunctive Scheduling. In *Proceedings of the Principles and Practice of Constraint Programming—CP 2015*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9255, pp. 393–402. [CrossRef]
39. Nowicki, E.; Smutnicki, C. An Advanced Tabu Search Algorithm for the Job Shop Problem. *J. Sched.* **2005**, *8*, 145–159. [CrossRef]
40. Van Laarhoven, P.J.M.; Aarts, E.H.L.; Lenstra, J.K. Job Shop Scheduling by Simulated Annealing. *Oper. Res.* **1992**, *40*, 113–125. [CrossRef]
41. Pardalos, P.M.; Shylo, O.V.; Vazacopoulos, A. Solving Job Shop Scheduling Problems Utilizing the Properties of Backbone and "Big Valley". *Comput. Optim. Appl.* **2010**, *47*, 61–76. [CrossRef]
42. Gonçalves, J.F.; Resende, M.G.C. An Extended Akers' Graphical Method with a Biased Random-Key Genetic Algorithm for Job-Shop Scheduling. *Int. Trans. Oper. Res.* **2014**, *21*, 215–246. [CrossRef]
43. Yuraszeck, F.; Mejia, G.; Rossit, D.A.; Lüer-Villagra, A. A Constraint Programming-Based Lower Bounding Procedure for the Job Shop Scheduling Problem. *Comput. Oper. Res.* **2025**, *177*, 106964. [CrossRef]
44. Yuraszeck, F.; Mejia, G.; Rossit, D.A.; Lüer-Villagra, A. A Note on "A Constraint Programming-Based Lower Bounding Procedure for the Job Shop Scheduling Problem". *Comput. Oper. Res.* **2026**, *189*, 107396. [CrossRef]
45. Shylo, O.V.; Shams, H. Boosting Binary Optimization via Binary Classification: A Case Study of Job Shop Scheduling. *arXiv* **2018**, arXiv:1808.10813.
46. Prud'homme, C.; Fages, J.G.; Lorca, X. Choco Documentation. 2024. Available online: https://choco-solver.org/ (accessed on 1 May 2026).
47. Schulte, C.; Tack, G.; Lagerkvist, M.Z. Modeling and Programming with Gecode. *Schulte Christ. Tack, Guid. Lagerkvist Mikael* **2010**. Available online: https://openreview.net/pdf?id=XODSB3NNTN (accessed on 11 June 2026).
48. Nethercote, N.; Stuckey, P.J.; Becket, R.; Brand, S.; Duck, G.J.; Tack, G. MiniZinc: Towards a Standard CP Modelling Language. In *Proceedings of the Principles and Practice of Constraint Programming—CP 2007*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4741, pp. 529–543. [CrossRef]
49. Beck, J.C. Solution-Guided Multi-Point Constructive Search for Job Shop Scheduling. *J. Artif. Intell. Res.* **2007**, *29*, 49–77. [CrossRef]

50. Meng, L.; Lu, C.; Zhang, B.; Ren, Y.; Lv, C.; Sang, H.; Li, J.; Zhang, C. Constraint Programming for Solving Four Complex Flexible Shop Scheduling Problems. *IET Collab. Intell. Manuf.* **2021**, *3*, 147–160. [CrossRef]
51. Taillard, É. Benchmarks for Basic Scheduling Problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]
52. Demirkol, E.; Mehta, S.; Uzsoy, R. Benchmarks for Shop Scheduling Problems. *Eur. J. Oper. Res.* **1998**, *109*, 137–141. [CrossRef]
53. Fisher, H.; Thompson, G.L. Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. *Ind. Sched. Prentice Hall.* **1963**, 225–251. Available online: https://www.scirp.org/reference/referencespapers?referenceid=1767744 (accessed on 11 June 2026).
54. Lawrence, S.R. *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*; Graduate School of Industrial Administration, Carnegie-Mellon University: Pittsburgh, PA, USA, 1984.
55. Adams, J.; Balas, E.; Zawack, D. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Manag. Sci.* **1988**, *34*, 391–401. [CrossRef]
56. Storer, R.H.; Wu, S.D.; Vaccari, R. New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling. *Manag. Sci.* **1992**, *38*, 1495–1509. [CrossRef]
57. Yamada, T.; Nakano, R. A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems. In *Proceedings of the Parallel Problem Solving from Nature 2 (PPSN-II)*; Elsevier: Amsterdam, The Netherlands, 1992; pp. 281–290.
58. Applegate, D.; Cook, W. A Computational Study of the Job-Shop Scheduling Problem. *Tech. Rep. Orsa J. Comput.* **1991**, *3*, 85–176. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.