Accelerating Constraint Programming Solver with Parallel External Heuristics: Experiments on Scheduling and Routing Problems

Vilém Heinz
Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic
Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Czech Republic

Šimon Zvára
Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Vít Knobloch
Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Zdeněk Hanzálek
Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Czech Republic

Petr Vilím
ScheduleOpt, Novy Knin, Czech Republic

Abstract

Constraint Programming (CP) is a powerful optimization method that provides optimality guarantees, but due to its exact nature, its scalability to large instances is often limited. To address this, we propose a hybrid approach that combines a CP solver with heuristic methods, directly and asynchronously exchanging solutions and objective values during runtime. The hybrid parallel configuration yields faster convergence to good solutions across various problem domains than the solver alone, while retaining the ability to guarantee optimality, leveraging the complementary nature of the CP solver and heuristics. The efficiency of this approach is evaluated on three well-known scheduling problems and three well-known routing problems. Noticeable improvements are observed for the Flow-shop Scheduling Problem (FSSP), the Traveling Salesman Problem (TSP), and the Vehicle Routing Problem with Time Windows (VRP-TW). Even for problems where improvements are marginal, the portfolio of methods increases the robustness of the approach.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Theory of computation → Discrete optimization; Mathematics of computing → Solvers; Mathematics of computing → Mathematical optimization; Mathematics of computing → Nonlinear equations; Mathematics of computing → Evolutionary algorithms; Applied computing → Industry and manufacturing

Keywords and phrases Constraint Programming, Hybridization, Heuristics, Discrete Optimization, Scheduling, Routing

Digital Object Identifier 10.4230/LIPIcs.CP.2026.28

Funding This work was co-funded by the European Union under the ROBOPROX project (reg. no. CZ.02.01.01/00/22_008/0004590), by the Grant Agency of the Czech Republic under the Project GACR 25-17904S and by the Czech Technical University in Prague, grant No. SGS25/144/OHK3/3T/13.

1 Introduction

In previous decades, operations researchers have proposed and refined many exact and heuristic methods to a wide variety of combinatorial problems. While exact approaches can obtain an optimal solution if sufficient time is available, heuristic approaches focus mainly on short runtime, providing solutions with loose or no quality guarantees. Thus, obtaining high-quality solutions with provable guarantees within practical time limits remains challenging for many combinatorial problems, particularly for industrial-scale instances.

Since good solution quality and short runtime demand different solution methods, the goal is to improve the ability of an exact solution method, specifically a Constraint Programming (CP) solver, to find good solutions quickly by cooperating with heuristics. Although techniques like warm-starting of exact methods with heuristic solution or iterative approaches using exact and heuristic algorithms to refine the solution are widely and successfully used, research focusing on full real-time parallel cooperation between these algorithms is much more limited.

Thus, the aim of this work is to propose such close cooperation and to leverage the strengths of the CP solver and heuristic algorithms to obtain the best possible solution in any given time. While our primary aim is to accelerate the CP solver, the cooperation works in both directions: for instance, heuristics can narrow the solver's search space, while the solver can in turn provide diverse solutions that help the heuristics escape local optima.

In particular, we use the OptalCP solver [53] and extend it with a communication interface enabling direct real-time integration of heuristics into the solution process. This capability is not available in existing exact solvers such as the state-of-the-art CP Optimizer [34], OR-Tools [24], Gurobi [26], Gecode [20], Chuffed [10], and Choco [9]; see Section 2.2 for more information.

Although OptalCP, as any other CP solver, is a combination of multiple algorithms, for the purposes of this paper, we will treat it as a single black-box component. The hybrid approach proposed in this paper combines the solver with metaheuristics and local search, making the formulation of the CP problem and the representation of the metaheuristic components the only problem-specific parts. By using metaheuristics and fairly generic local search rather than problem-tailored heuristics, we aimed to reduce the effort required to address multiple different problems at once.

The efficiency of the proposed hybrid approach is evaluated using three prominent scheduling problems and three prominent routing problems. The results show substantial improvements on three of the six problems, both in the quality of the solutions achieved within a fixed time limit and in the speed with which solutions of particular quality are obtained using the same computational resources. The hybrid approach further increases robustness by compensating for weaknesses of either the CP solver or of the heuristics, while providing optimality-gap guarantees.

1.1 Contribution and Outline

The main contributions of this paper are:

1. A novel parallel and fully asynchronous hybrid approach using the OptalCP solver and external heuristics, including the development of a communication interface that supports real-time solution exchange, enabling active cooperation between methods.
2. A practical black-box integration framework that allows the use of external solutions methods without modifying the OptalCP solver, making the hybrid design problem-agnostic aside from the CP model and the heuristic representation.

3. A broad empirical analysis across six benchmark problems, identifying and analyzing when hybridization provides noticeable benefits (FSSP, TSP, VRP-TW) versus when the solver alone is sufficient (JSSP, RCPSP, CVRP).

The remainder of the paper is organized as follows. Section 2 reviews the existing literature on approaches that combine exact and heuristic methods. Section 3 describes the proposed architecture, the communication interface and the heuristic methods used. Section 4 presents the computational evaluation, comparing the hybrid approach with the OptalCP and CP Optimizer references. Finally, Section 5 summarizes the paper and discusses possible future research.

2 Related Work

Although existing surveys [6, 56, 70, 17] on exact and metaheuristic hybridization mention many approaches, most of them are either solely combinations of metaheuristics, or limit how exact and metaheuristic algorithms cooperate. To illustrate the differences between existing hybrid combinations, we can categorize them using the classification hierarchy proposed in the survey paper [56] and provided in Figure 1.

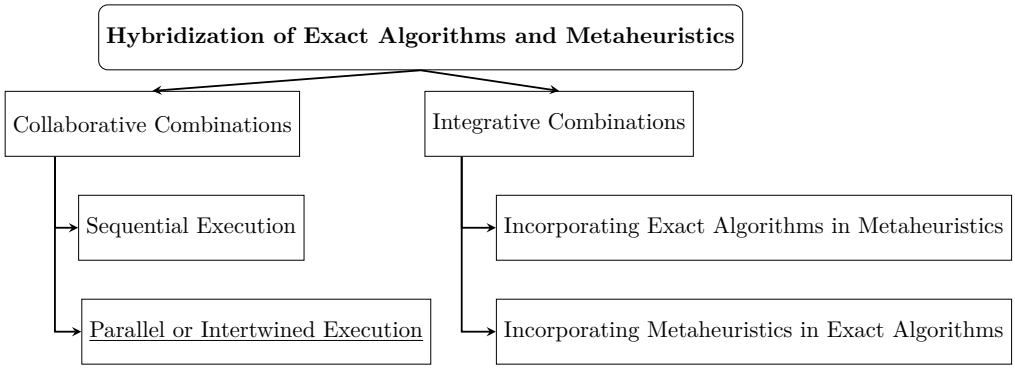

Figure 1 Hierarchy of hybrid combinations based on cooperation type. The underlined text denotes the category of our hybrid approach. Adapted from [56].

At the highest level, the hierarchy distinguishes between integrative and collaborative combinations. Integrative combinations (the right branch) are approaches in which one algorithm is a subordinate part embedded in the other, usually solving some subproblem. An example of this can be the heuristic estimation or the guiding of computationally expensive steps of an exact algorithm. On the other hand, collaborative combinations (the left branch) are approaches in which the algorithms are on the same level, meaning none are contained in another. For sequential combinations, this means that the algorithms are executed one after the other. A typical example of this is warm-starting, where a heuristic provides an initial solution for the exact algorithm. However, the sequential nature largely limits how the algorithms can interact. For parallel/intertwined combinations, this is not the case, as the algorithms are executed side by side and are able to exchange and utilize the information produced by others throughout the solving runtime.

Thus, the related work is organized into two subsections. Subsection 2.1 reviews collaborative sequential and integrative combinations. Since these are not closely related to our approach, we restrict the discussion to works that involve CP, allowing for a comparison

to our approach in terms of functionality. Subsection 2.2 then examines collaborative parallel combinations (underlined in Figure 1), which constitute our main focus, as they are particularly relevant to our approach.

2.1 Sequential and Integrative Combinations Using CP

A very popular sequential approach to hybridizing CP with heuristics is the aforementioned warm-starting. The authors of [31] propose an example of CP warm-starting, providing the best of multiple heuristic solutions as the initial solution used by the CP solver for the problem of parallel machine scheduling. The results demonstrate that the solver is noticeably accelerated and is able to reach much better solutions in the same time limit.

However, CP can also be used as the method providing the warm-start. In [25], a highly constrained periodic scheduling problem is introduced and CP is used to initialize the local search method due to the difficulty in obtaining a feasible starting solution in such a constrained scenario.

The authors of [37] proposed a more complex variant of warm-start, called *hot start*. In this case, the authors used Ant Colony Optimization (ACO) pheromone trails to first sample feasible solutions by CP, and then the resulting pheromone matrix is used to navigate the exhaustive search within CP in the second phase, improving its convergence. The use of pheromone matrix in the exhaustive search provides a particular advantage over the usual warm-start use.

A different use of sequential solution passing are sequential CP portfolios. In [49], the authors propose something called CPHydra which is a case-based reasoning which essentially uses the known past performance on similar cases (instances) to determine which combinations of solvers and in what order they should be executed. A nice extensive empirical study on the portfolio performances of various existing solvers is provided by [3]. A related sequential strategy is proposed in [2] for the multimode Resource-Constrained Project Scheduling Problem, where the CP and LNS algorithms are alternated and an additional perturbation step is applied whenever no improvement is found. The approach demonstrates improved lower bounds on established benchmark instances.

Moreover, CP and heuristics can also be combined integratively within a single algorithmic framework. One such approach combining ACO and CP is proposed in [45]. ACO serves as the main procedure, while CP is used by the individual ants as a tool to construct solutions. Therefore, if ants use CP to find a feasible solution, the search focuses on finding a good-quality solution among the feasible ones, and a great deal of computational effort can be saved. In other words, CP provides a filtering mechanism, while ACO comes into focus in the context of variable/value selection.

Another practical integrative application of CP is in problem decomposition. For example, [62] employs CP to address a subproblem within a larger solution framework that combines CP, MILP, and a decomposition heuristic. CP is specifically used to handle a scheduling subproblem, leveraging its efficiency and suitability for this type of task.

The collaborative sequential combination approach most closely related to this paper is proposed by the authors of [23]. A combination of three algorithms, Simulated Annealing (SA), Tabu Search (TS), and Constraint Logic Programming (CLP), is used. While SA is the main algorithm, CLP is used to exhaustively explore the search space of proposed subproblems, and TS to prevent cycling in the same part of the solution space. However, compared to our approach, their proposed method does not provide optimality guarantees, as CLP is only used to exhaustively explore subproblems, and it is not parallel, since the local search part (comprising SA and TS) operates in an alternating fashion with the CLP.

Lastly, we mention the CP-based Large Neighborhood Search (CP-LNS). Today, it is an integral part of CP solvers such as OptalCP [53], CP Optimizer [34], and Google OR-Tools [24]. The general idea is to use LNS to define a neighborhood around the current solution and then apply constraint programming to search for improved solutions. This technique was introduced in [65] and later refined in [66] and [51]. Its adaptation to single-mode scheduling problems is described in [41]. The use of CP-LNS in OptalCP is discussed in Section 3.

2.2 Collaborative Parallel Combinations

As stated in the survey [56], the research on collaborative parallel combinations is much less prominent compared to the other combinations. This is likely due to the need for asynchronous communication and efficient use of the data exchanged at any step of the solution process by all algorithms. Although such designs are more challenging to implement than collaborative sequential or integrative approaches, they offer a key advantage: information exchange is not restricted to predefined points in the process, nor are the algorithms bound to rigid roles such as master and subroutine. Consequently, collaborative parallel combinations provide greater flexibility and the potential for higher efficiency.

We start by reviewing the approaches to parallelism in CP, in particular portfolio-based methods. Although these works do not combine the exact solver with external heuristics and offer limited or no collaboration between algorithms, they are relevant due to their use of CP algorithms in parallel settings. An extensive experimental comparison of portfolio-based approaches and search-space splitting is presented in [7]. The results demonstrate that even independent workers without runtime information exchange can achieve substantial performance gains.

An approach that enables runtime exchange of solution bounds within a portfolio of existing CP solvers is proposed in [4]. However, since the solvers used cannot incorporate improved incumbents during search, exploiting new bounds requires restarting the worker, thus losing current search state information. An approach that allows exchange of bounds and/or learned clauses during runtime is proposed in [18], with the authors concluding that sharing of bounds is vital for efficient parallel search. Nevertheless, this cooperation is integrated into the solver architecture and does not provide a general black-box mechanism for combining heterogeneous approaches. Similarly, [28] considers a portfolio of backtracking CSP algorithms that exchange information such as supports or no-goods but only within the internal search architecture.

We should also briefly mention other approaches to CP parallelism that focus on methods such as splitting into subproblems [46], [58], or parallelization strategies such as restart-based diversification [11], which are, however, conceptually different from our setting. A comprehensive overview of parallel CP techniques can be found in [57] (Chapter 9) and [29]. In summary, while some CP portfolio architectures support exchanging bounds, no-goods, or other partial information, there is no general black-box CP framework that allows real-time asynchronous injection of complete solutions from external sources into an independently running CP solver without restart or solver modification.

Beyond CP-specific approaches, several general collaborative optimization frameworks have also been proposed. In [71], the authors propose a collaborative parallel combination called A-Teams. A-Teams is a combination of independent agents that solve the same problem and exchange data, e.g., solutions, through shared memory. While it does not enforce the use of specific algorithms, two types of algorithms are defined; constructive, providing new solutions, and destructive, removing unfit solutions from the consideration of other algorithms. This framework was, for example, applied by the authors of [5] to the job-shop scheduling problem, demonstrating its practical use.

Table 1 Comparison of existing approaches to ours. [1]A-Teams rely on shared-memory communication. [2]The BnB+TS cooperation depends on method-specific internals.

Approach	(i) Async	(ii) Exact	(iii) Black-box	(iv) General
A-Teams [71]	Yes	No	Yes[1]	No
TECHS [16]	No	Yes	No	No
BnB+TS [33]	Yes	Yes	No[2]	No
This paper	Yes	Yes	**Yes**	**Yes**

Another parallel combination called TECHS was proposed by the authors of [16]. TECHS is also a multi-agent-based approach where agents exchange information to enhance their search. However, unlike A-Teams, where solutions are continuously exchanged, TECHS employs a structured approach in which agents periodically stop their search to exchange filtered information. The information flow is regulated by referees who decide which solutions should be shared. TECHS application is demonstrated in the same paper, using the combination of Branch-and-Bound (BnB) and Genetic Algorithm to solve the job-shop scheduling problem.

An approach that specifically combines an exact algorithm with multiple metaheuristics workers was proposed in [33]. Specifically, the authors used a BnB algorithm combined with Tabu Search (TS). The BnB uses solutions provided by TS workers to cut suboptimal branches, and if any TS worker gets stuck, it generates a new initial solution that lies in the unexplored space of the BnB. The results show that the hybrid variant is much more performant than using the algorithms separately.

To clearly distinguish the most relevant parallel approaches in the literature from the one proposed in this paper, we present Table 1. Papers focusing solely on CP-based parallel methods, such as CP portfolios, are omitted from the table because our focus is on combining exact and heuristic approaches, and these methods do not support cooperation with other external algorithms. As can be seen in the table, our approach unifies all previously separated aspects in one: (i) fully asynchronous all-to-all communication, (ii) the ability to prove optimality, (iii) black-box integration of external user-designed solution methods to existing OptalCP solver, and (iv) the use of a general-purpose exact approach. To the best of our knowledge, no prior work has proposed and empirically validated the efficiency of such combination of general-purpose exact approach and (meta)heuristics that meets all the specified criteria.

Lastly, we discuss the limitations of existing exact solvers, to further demonstrate the novelty of our approach. In Section 1, we mentioned CP Optimizer [34], Gurobi [26], OR-Tools [24], Gecode [20], Chuffed [10], and Choco [9]; not all are (purely) CP solvers. While these solvers allow the user to provide a hint (initial solution) before starting the search, real-time parallel cooperation using an externally provided solution is limited or not possible at all. CP Optimizer offers callbacks (the ability to react to events that occur during the search, such as a newly found solution), but they are essentially observational and do not allow modification of the ongoing search or injection of new incumbents. Gurobi allows externally generated solutions to be submitted during runtime, but only at specific callback points and the processing of the solution can be delayed, so solutions cannot be injected and used at arbitrary times. Although OR-Tools, Gecode, Chuffed, and Choco are open source, they do not provide an existing documented mechanism for injecting new solutions once the search has started. Consequently, in most solvers, providing an externally generated solution that improves the current incumbent requires restarting the search, which results in the loss

of part or all of the accumulated internal knowledge. To the best of our knowledge, this makes our approach the first that enables real-time asynchronous cooperation between a general-purpose exact solver and external algorithms without any practical limitations.

3 Hybrid Architecture and Heuristic Components

The hybrid approach proposed in this paper is made up of two main parts. The first part is the OptalCP CP solver that contains two search algorithms: Failure-Directed Search (FDS) [73, 30] and Large Neighborhood Search (LNS), similar to the one proposed in [41] (also referred to as CP-LNS in the previous section). The second part is the heuristic portfolio that consists of all (meta)heuristics used. Note that the descriptions of specific heuristics used provided in Section 3.2 and Section 3.3 remain high-level, as they can be replaced by arbitrary different ones. The instances of particular heuristics or internal solver search algorithms are called *workers*; the types of particular OptalCP workers are controlled by the `SearchType` parameter. The heuristic workers and the solver's manager, called *Master*, asynchronously exchange newly found solutions through the Node.js interface. A high-level illustration of the hybrid approach is presented in Figure 2.

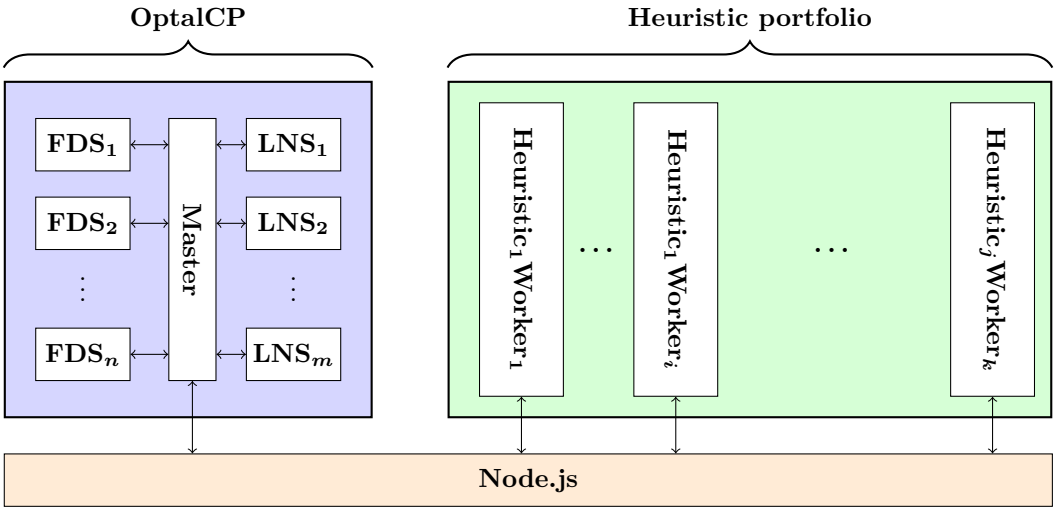

Figure 2 The structure of the hybrid approach.

3.1 Communication Interface

The critical part of the proposed approach is the fast and asynchronous exchange of solutions between various algorithms. The solver and external heuristics run as separate processes, orchestrated by a lightweight Node.js application that forwards solutions between them through newline-delimited JSON messages over standard input and output. This design is language-agnostic, meaning that any programming language can serve as a heuristic, as long as it can read and write the simple JSON solution format. Our experiments use the Node.js API, but OptalCP also provides a Python API with equivalent functionality. The heuristics themselves are written in C++.

The OptalCP API provides a callback-based interface that allows the user to register a function invoked each time the solver finds a new solution. The user can also call `Solver.sendSolution` to asynchronously deliver an external solution to the solver during the search. Figure 3 shows a usage example: the JavaScript glue code (i.e., the user-written

code that connects OptalCP with an external heuristic). Thanks to the API design, the code is minimal: it only translates solutions between the two formats and forwards them. Other tasks such as solution verification, distribution to workers, and search space pruning are handled internally by the solver. A complete demo is available on GitHub [52].

```
import * as CP from '@scheduleopt/optalcp'
...
let solver = new CP.Solver;
let heuristic = spawn(...); // Start heuristic subprocess
solver.onSolution = (event) => { // When solver finds a solution
  heuristic.stdin.write(toHeuristicFormat(event.solution));
};
heuristic.stdout.on('data', (data) => { // When heuristic finds one
  solver.sendSolution(fromHeuristicFormat(data));
});
await solver.solve(model, parameters);
```

Figure 3 Communication between OptalCP and an external heuristic. Functions `toHeuristicFormat` and `fromHeuristicFormat` translate solutions between the OptalCP and heuristic representations.

When an external solution arrives, it enters a dedicated reader thread that parses the JSON message and queues it for the Master. The Master first verifies the solution against all model constraints (controlled by the `verifyExternalSolutions` parameter, enabled by default). Invalid solutions are rejected and logged, while valid solutions that do not improve on current best are discarded. If the solution improves the best known objective, the Master distributes it to all internal workers through lock-free queues, one per each worker. This distribution is asynchronous, meaning that no worker is interrupted or blocked.

Each worker independently polls its queue for new solutions at the smallest possible natural synchronization points in its algorithm. The tree search (FDS in particular) checks for incoming solutions at every backtrack: when a better solution arrives, its objective value tightens the pruning bound, allowing the worker to cut branches that cannot lead to improvements. LNS checks at the beginning of each iteration: if a better solution has arrived, it replaces the current incumbent used for destruction and repair. In both cases, external solutions are treated identically to solutions found internally (the workers do not distinguish between the two origins).

The latency of `sendSolution` (from the API call until the solver processes the solution, including verification) is approximately 1–1.5 ms for models with up to 2 500 variables, which covers the majority of our benchmark instances and is negligible compared to typical solve times. Heuristics should, nevertheless, filter solutions and only send ones that improve upon previously sent values. While the solver handles duplicates gracefully, processing and verifying redundant solutions still consume time, especially when many arrive in rapid succession.

Overall, the communication interface treats the solver as a black box from the heuristic's perspective. Heuristic developers do not need to understand constraint propagation, search strategies, or solver internals; they only need to produce valid solutions in the expected format. This makes it easy to reuse existing heuristic implementations written in any programming language. The solver automatically manages all coordination: best solution tracking, objective bounds, optimality gap computation, and termination when optimality is proved. External heuristics simply send and receive solutions without needing to be aware of each other. This enables a cascade of improvements, where a solution found by one algorithm is improved by another, then further refined by a third, and so on.

3.2 Metaheuristics for Scheduling Problems

Scheduling problems are a typical use case for CP, as they generally belong to the $\mathcal{NP}$-hard complexity class and have very natural CP formulations. We considered three well-known problems: the Flow-shop Scheduling Problem (FSSP) [35], the Job-shop Scheduling Problem (JSSP) [44], and the Resource-Constrained Project Scheduling Problem (RCPSP) [55].

For all three problems, we used the same metaheuristic algorithms: Genetic Algorithm (GA), Large Neighborhood Search (LNS), Particle Swarm Optimization (PSO), Simulated Annealing (SA), and Tabu Search (TS). The selection was based on popular and successful methods from the survey study [50] studying efficient methods for one of our measured problems, RCPSP. The implementations were mainly inspired by the following works: GA [54, 19], LNS [43, 76], PSO [36, 74], SA [39, 8], and TS [21, 64]. The core of each metaheuristic and its components are reusable, with the core remaining the same for every scheduling problem, while the components/operators are adjusted if required. When a new improving solution is found by any worker, population-based methods (GA and PSO) add this new solution as an individual to their population, while local search-based methods (SA, LNS and TS) replace their current one.

3.3 Metaheuristics and Heuristics for Routing Problems

Routing problems are another important class of $\mathcal{NP}$-hard problems. Since routing problems are less of a typical application for CP, we used them to see how it affects the usefulness of hybridization in comparison to scheduling problems. We considered three well-known problems: Travelling Salesman Problem (TSP) [61], Capacitated Vehicle Routing Problem (CVRP) [13], and Vehicle Routing Problem with Time Windows (VRP-TW) [67].

For each of the problems, we implemented two generalizable heuristics. Highly specialized algorithms, such as LKH [32] designed specifically for TSP, were not implemented because they would not easily generalize to other problems. For TSP, a local search with 2-OPT perturbation [22] and a genetic algorithm with 2-OPT mutation [22] and PMX crossover [27] were implemented. For CVRP, local search with 2-OPT, exchange and relocation perturbation operators [38] and stochastic ranking genetic algorithm (SRGA) [63] with the same operators [38], and PMX crossover [27] were implemented. For VRP-TW, the methods and operators used were essentially the same as in CVRP, but adapted to respect the additional time-window constraints. Since the time windows constraints are significantly different from TSP and CVRP, we decided to also introduce a dedicated memetic algorithm [47] in this case. This algorithm combines selection, PMX crossover, and a relocation-style mutation, followed by an improvement phase in which the four operators (2-opt, relocate, swap, and cross) are applied to refine offspring solutions. To clearly summarize the configurations of all routing heuristics, we provide Table 2. Similarly to scheduling metaheuristics, a new improving solution found by any worker is inserted into the population in case of GA (and memetic algorithm in VRP-TW), while replacing the current solution in the local search method.

Table 2 Routing heuristics used.

Problem	Heuristic 1	Heuristic 2	Heuristic 3
TSP	LS (2-OPT)	GA (2-OPT mut., PMX)	–
CVRP	LS (2-OPT, exch., reloc.)	SRGA (exch., reloc., PMX)	–
VRP-TW	CVRP LS (adapted)	CVRP SRGA (adapted)	Memetic (see text)

4 Experimental Results

As stated in the previous sections, the aim of our hybrid method is to improve the solver's capabilities and to enhance overall efficiency by leveraging the complementary strengths of both approaches. An example of this successful cooperation is shown in Figure 4. The improved solutions obtained during the solving process for a given JSSP instance are depicted as points, with time on the x-axis and the objective value on the y-axis; the point's color indicates whether the improvement was achieved by a heuristic (rectangle) or a solver (dot) worker.

As expected, heuristic workers provide most of the fast good solutions moving the search at the start, while the solver takes the initiative later. However, heuristic still contributes in the later stage of the search, as solver's systematic exploration moves the search out of the local optima providing new opportunities for improvement – as demonstrated by the heuristic solution found at time 80. Moreover, the dashed and dotted lines, showing results obtained with the same time limit and number of workers using only the solver or only metaheuristics, respectively, indicate that even their combined 24 workers without solver-metaheuristics communication would yield a worse result. This indicates that exchanging solutions between solver and heuristics during runtime in this particular instance maximizes the use of the complementary nature of the methods; an advantage over a non-communicating portfolio of concurrent algorithms.

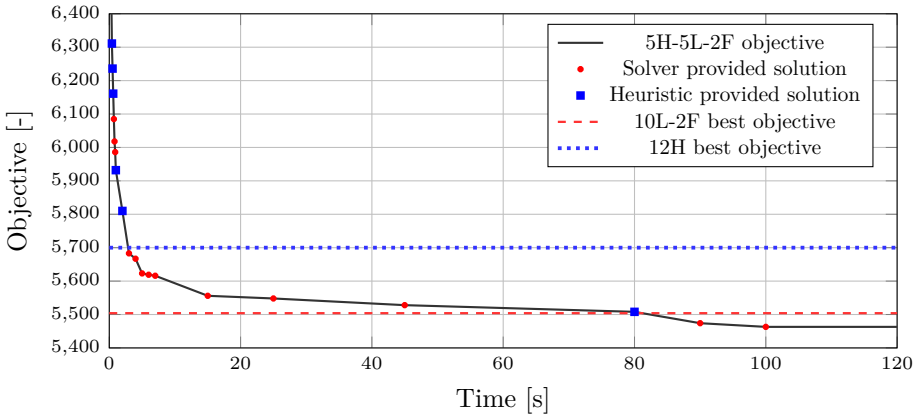

Figure 4 Evolution of best solution found by a hybrid configuration containing 5 heuristic workers and 7 OptalCP ones (5H-5L-2F) on JSSP instance *cscmax_40_15_7*. The dashed and dotted horizontal lines represent solutions found in the complete 120-second time limit using 12 workers of OptalCP (10L-2F) and 12 heuristic workers (12H), respectively.

4.1 Evaluation Methodology

Before presenting the experimental results, we clarify why we do not compare our method with existing approaches. This is largely due to two reasons: (i) as discussed in Section 2, there exist no closely related approaches to the one proposed in this paper, and (ii) the objective of experiments is not to compare with state-of-the-art methods for tested problems, but rather to evaluate the potential of the proposed hybridization.

To summarize the results obtained on the experimental datasets, we used the following two methods. The first is the contour of the Runtime Distribution Function (RDF) [60]. The RDF contour is a function of time (x-axis) and relative solution quality achieved by a given

approach compared to the best solution found (y-axis). The best solution could be obtained by any approach of the measured ones, at any time (up to the time limit), in any run (e.g., random seed). Because some of the measured approaches may not find a feasible solution on all instances within a given time, we report the 90th percentile of the relative solution quality, averaged over all seeds and instances (runs). Thus, the curve begins only once at least 90% of the runs have found a feasible solution. In other words, the RDF contour represents an anytime gap, but not to the optimal solution (which we often do not have), but to the best solution objective attained.

The second method is a bar chart, in which the objectives of all instance solutions obtained by two different approaches are compared individually. For each instance, the relative difference between two objective values A and B is calculated as $(A - B)/\max(A, B)$ and used to sort the instances to demonstrate performance differences.

For the evaluation of scheduling problems, we used 60 largest Taillard instances [69] for FSSP, 100 randomly selected instances from standard datasets [1, 68, 69, 15, 75] for JSSP and 100 randomly selected RG300 instances [14] for RCPSP. For routing problems, we used TSPLIB [59] instances for TSP (with the exception of the few largest instances), Test Set A of CVRP Lib [72] for CVRP, and Solomon test set [67] instances with 100 customers for VRP-TW. Each instance was executed with three different random seeds for every configuration using a 120-second time limit. The official OptalCP repository containing all the benchmarks used is available at [52].

For every problem, three different configurations were tested, each using 12 threads to provide a fair comparison: (i) OptalCP with 10 LNS and 2 FDS threads (10L-2F), (ii) hybrid approach combining one thread for each implemented (meta)heuristic with 2 FDS threads and the remaining threads allocated to LNS (5H-5L-2F for scheduling and 2H-8L-2F/3H-7L-2F for routing), and (iii) CP Optimizer solver with default configuration using 12 threads (12CPO). CP Optimizer was selected as a benchmark reference because it is a state-of-the-art CP solver, especially effective in scheduling problems [42, 40, 48, 12], where it is often able to keep up with heuristics even in short time limits. In Table 3, we provide a brief overview of the OptalCP hybrid configurations and the datasets used.

Table 3 OptalCP hybrid configurations and respective datasets used for each problem.

Problem	Config.	Instances	Selection	Source
FSSP	5H-5L-2F	60	largest	Taillard [69]
JSSP	5H-5L-2F	100	random	Mixed [1, 68, 69, 15, 75]
RCPSP	5H-5L-2F	100	random	RG300 [14]
TSP	2H-8L-2F	84	excl. largest	TSPLIB [59]
CVRP	2H-8L-2F	27	all	CVRP Lib Set A [72]
VRP-TW	3H-7L-2F	56	100-customer	Solomon [67]

All experiments were run on a server with a Ryzen 7700X processor and 64GB of RAM. The results of scheduling problems are shown in Figure 5 and of routing problems in Figure 6. The left column represents a comparison of OptalCP, hybrid approach, and CP Optimizer; the right column represents a per-instance comparison of hybrid approach and OptalCP. The model formulations of the problems considered by OptalCP and CPOptimizer were the same.

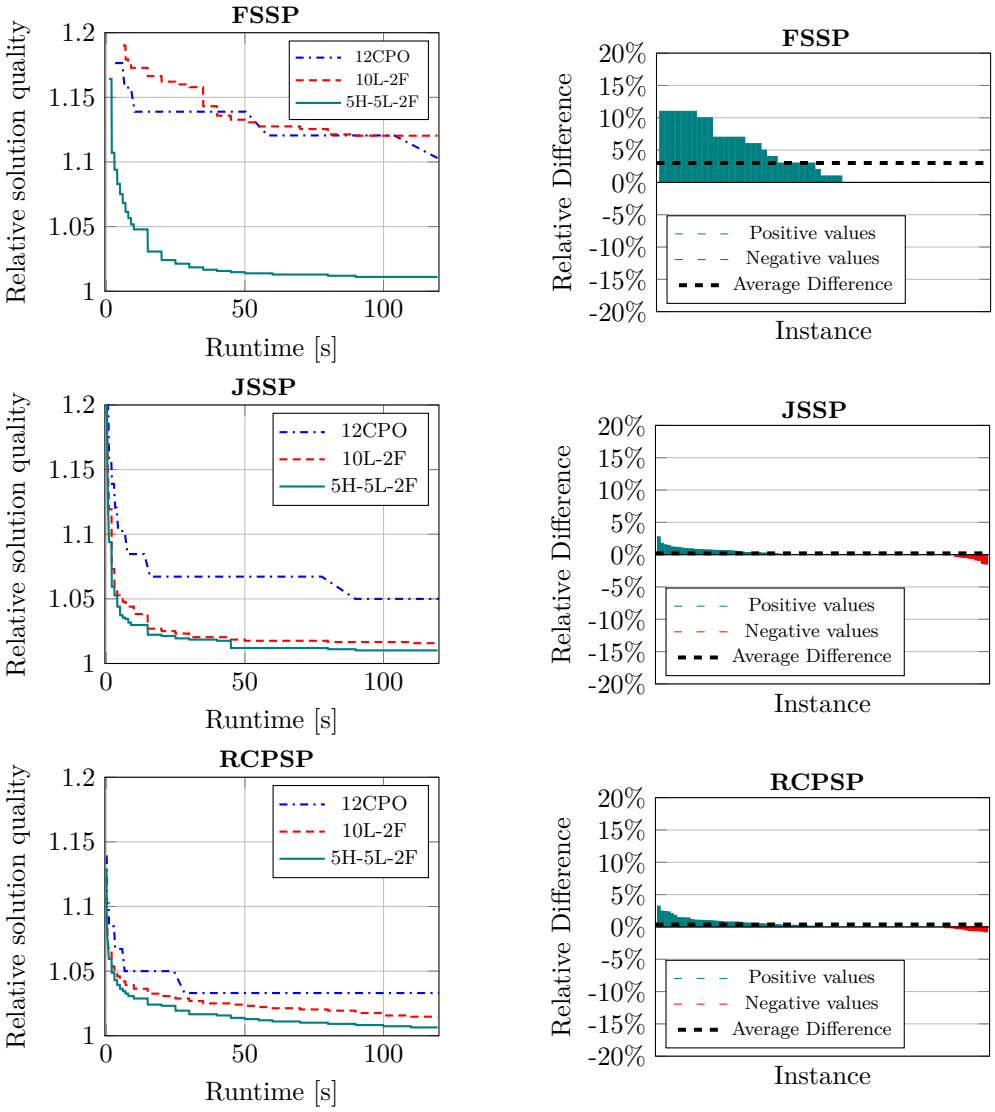

Figure 5 *Left column:* Contour plots comparing configurations 12CPO, 10L-2F, and 5H-5L-2F. *Right column:* Relative performance per instance; positive values indicates better performance of 5H-5L-2F (OptalCP with heuristics) and negative values better performance of 10L-2F (OptalCP only) configuration. All measurements done in 120-second time limit.

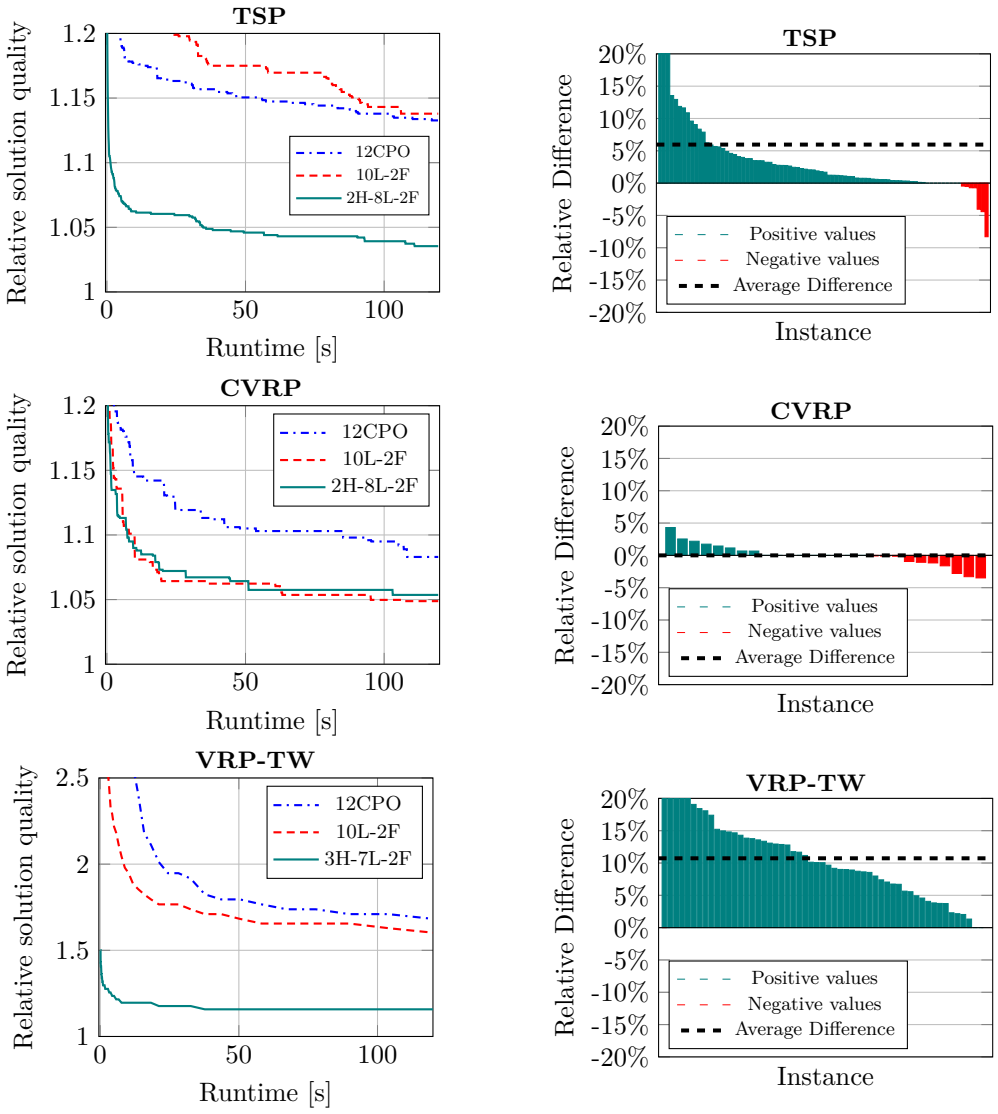

Figure 6 *Left column:* Contour plots comparing configurations 12CPO, 10L-2F, and 2H-8L-2F/3H-7L-2F. *Right column:* Relative performance per instance; positive values indicates better performance of 2H-8L-2F/3H-7L-2F (OptalCP with heuristics) and negative values better performance of 10L-2F (OptalCP only) configuration. Please note that in VRP-TW contour, there is differently scaled y-axis. All measurements done in 120-second time limit.

4.2 Results Discussion

The most favorable results were obtained for the FSSP, TSP and VRP-TW problems, with average improvements per instance of 3%, 6% and 11%, respectively. Moreover, the contour plots for these problems show that good solutions were generally found very early by the hybrid approach, yielding even larger relative gains in shorter time limits. The bar charts further indicate that almost all instances benefited from the hybridization (teal positive-value instances), with just a few TSP instances measurably worsening. For the other scheduling problems (JSSP and RCPSP), the hybrid configuration performed slightly better on average, while no improvement was achieved in CVRP. When considering all six problems, the hybrid approach solved 60% of the instances better while only 10% worse than the OptalCP solver.

We attribute the results in JSSP, RCPSP, and CVRP to the fact that the solver alone already demonstrates strong performance in them, especially in JSSP and RCPSP. This is evidenced by (i) the fact that OptalCP solver outperforms the state-of-the-art CP Optimizer in all problems except FSSP and (ii) a recent paper [30], where OptalCP achieves new state-of-the-art lower bounds on hundreds of JSSP and RCPSP standard instances while closing a few of them in just a 900-second time limit. Furthermore, the heuristics used for all problems were initially developed and fine-tuned for FSSP and TSP respectively, which likely explains their stronger performance there. For VRP-TW, we included an additional problem-specific memetic algorithm, which turned out to be the most effective among the heuristics used for VRP-TW and explains the fast convergence seen in its contour plot. Consequently, the limited gains in JSSP, RCPSP, and CVRP problems are an expected outcome: when the solver is already strong on a given problem, hybridization requires equally strong, problem-aware heuristics, to contribute more meaningfully.

Lastly, even in cases where average improvements are modest or absent, hybridization still contributes to overall robustness. For example, in three TSP instances, the solver alone failed to even construct a feasible solution within the 120-second time limit, while the availability of heuristic solutions enabled the solving process to proceed. On the other hand, heuristic solutions are frequently improved by the solver later in the solving process, with the heuristic still contributing new solutions due to the search space guidance of the CP solver; as illustrated in Figure 4.

5 Conclusion and Outlook

This paper presents a novel hybrid approach that combines Constraint Programming (CP) with a portfolio of (meta)heuristic algorithms cooperating in an asynchronous parallel manner. A general framework and communication interface are proposed that allow information exchange between the CP solver OptalCP and external solution methods. Several established metaheuristics from the literature are implemented and used in the hybrid approach.

Extensive experiments demonstrate noticeable improvements in three of the six well-known scheduling and routing problems compared to the solver alone, improving solution quality and convergence speed while using the same computational resources. The experiments also demonstrate a strong overall performance of OptalCP by comparing it with the state-of-the-art CP solver CP Optimizer, explaining weaker results in the remaining three problems and showing that sufficiently powerful heuristics are required to contribute meaningfully in problems where the solver already performs well. The hybrid approach also provides implicit robustness against adversarial instances for any given particular algorithm, demonstrated in a few instances where the solver alone failed to produce any solution within a given time limit. Additionally, the ability to provide optimality gaps gives our method a practical edge because it provides quality assurances of the produced solution.

Future work includes focusing on information exchange beyond the solution and its objective value, adaptive online selection of heuristics, and incorporating learning-based components to guide search decisions. We would also like to investigate the benefits of real-time solution exchange in greater depth, particularly to determine how much of the improvement is attributable to the exchange itself and how much stems from portfolio diversity.

References

1 Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988. URL: http://www.jstor.org/stable/2632051.

2 Arben Ahmeti and Nysret Musliu. Hybridizing constraint programming and meta-heuristics for multi-mode resource-constrained multiple projects scheduling problem. *Journal of Heuristics*, 31(1):2, November 2024. doi:10.1007/s10732-024-09540-3.

3 Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. An empirical evaluation of portfolios approaches for solving CSPs, 2014. arXiv:1212.0692.

4 Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. A multicore tool for constraint solving. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 232–238. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/039.

5 M. Emin Aydin and Terence C. Fogarty. Teams of autonomous agents for job-shop scheduling problems: An experimental study. *Journal of Intelligent Manufacturing*, 15(4):455–462, August 2004. doi:10.1023/B:JIMS.0000034108.66105.59.

6 Christian Blum, Jakob Puchinger, Günther R. Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011. doi:10.1016/j.asoc.2011.02.032.

7 Lucas Bordeaux, Youssef Hamadi, and Horst Samulowitz. Experiments with massively parallel constraint solving. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 443–448, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc. URL: http://ijcai.org/Proceedings/09/Papers/081.pdf.

8 Shouvik Chakraborty and Sandeep Bhowmik. Job shop scheduling using simulated annealing. In *First International Conference on Computation and Communication Advancement*, January 2013.

9 Choco Team. Choco-solver: Java Constraint Programming Library. https://choco-solver.org/, 2026. Accessed: 2026-03-02.

10 Chuffed Development Team. Chuffed Constraint Solver. https://github.com/chuffed/chuffed, 2026. Accessed: 2026-03-02.

11 Andre Cire, Serdar Kadioglu, and Meinolf Sellmann. Parallel restarted search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), June 2014. doi:10.1609/aaai.v28i1.8848.

12 Giacomo Da Col and Erich Teppan. Google vs IBM: A constraint solving challenge on the job-shop scheduling problem. *Electronic Proceedings in Theoretical Computer Science*, 306:259–265, September 2019. doi:10.4204/eptcs.306.30.

13 G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959. URL: http://www.jstor.org/stable/2627477.

14 Dieter Debels and Mario Vanhoucke. A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55(3):457–469, 2007. doi:10.1287/OPRE.1060.0358.

15 Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998. doi:10.1016/S0377-2217(97)00019-2.

16 J. Denzinger and T. Offermann. On cooperation between evolutionary algorithms and other search paradigms. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 2317–2324 Vol. 3, 1999. doi:10.1109/CEC.1999.785563.

17 Suling Duan, Shanlin Jiang, Huan Dai, Luping Wang, and Zhenan He. The applications of hybrid approach combining exact method and evolutionary algorithm in combinatorial optimization. *Journal of Computational Design and Engineering*, 10(3):934–946, April 2023. doi:10.1093/jcde/qwad029.

18 Thorsten Ehlers and Peter J. Stuckey. Parallelizing constraint programming with learning. In Claude-Guy Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming: 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29–June 1, 2016, Proceedings*, volume 9676 of *Lecture Notes in Computer Science*, pages 142–158, Cham, Switzerland, 2016. Springer. Conference paper; Scopus ID: 84978643547. doi:10.1007/978-3-319-33954-2_11.

19 E. Falkenauer and S. Bouffouix. A genetic algorithm for job shop. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 824–829 vol.1, 1991. doi:10.1109/ROBOT.1991.131689.

20 Gecode Project. Gecode: Generic Constraint Development Environment. https://www.gecode.dev/, 2026. Accessed: 2026-03-02.

21 Michel Gendreau and Jean-Yves Potvin. *Tabu Search*, pages 165–186. Springer US, Boston, MA, 2005. doi:10.1007/0-387-28356-0_6.

22 David E. Goldberg and Robert Lingle. Alleles, loci, and the traveling salesman problem. *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, 1985.

23 Nuno Gomes, Zita A. Vale, and Carlos Ramos. Combining metaheuristics and constraint programming to solve a scheduling problem. In *AMCOS'05: Proceedings of the 4th WSEAS International Conference on Applied Mathematics and Computer Science*, 2005. URL: https://api.semanticscholar.org/CorpusID:17343176.

24 Google LLC. OR-Tools: Open Source Optimization Tools. https://developers.google.com/optimization/, 2026. Accessed: 2026-03-02.

25 Josef Grus, Claire Hanen, and Zdeněk Hanzálek. Periodic chains scheduling on dedicated resources – A crucial problem in time-sensitive networks. *Computers & Operations Research*, 180:107072, 2025. doi:10.1016/j.cor.2025.107072.

26 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. https://www.gurobi.com, 2026. Accessed: 2026-03-02.

27 Mais Haj-Rachid, Wahiba Ramdane-Cherif, Pascal Chatonnay, and Christelle Bloch. Comparing the performance of genetic operators for the vehicle routing problem. *IFAC Proceedings Volumes*, 43(17):313–319, 2010. 5th IFAC Conference on Management and Control of Production Logistics. doi:10.3182/20100908-3-PT-3007.00068.

28 Youssef Hamadi and Georg Ringwelski. Boosting distributed constraint satisfaction. *Journal of Heuristics*, 17(3):251–279, June 2011. doi:10.1007/s10732-010-9134-2.

29 Youssef Hamadi and Lakhdar Sais, editors. *Handbook of Parallel Constraint Reasoning*. Springer, Cham, 1 edition, 2018. doi:10.1007/978-3-319-63516-3.

30 Vilém Heinz, Zdeněk Hanzálek, and Petr Vilím. Reinforcement learning for search tree size minimization in constraint programming: New results on scheduling benchmarks. *SSRN*, 2024. doi:10.2139/ssrn.4938242.

31 Vilém Heinz, Antonín Novák, Marek Vlk, and Zdeněk Hanzálek. Constraint programming and constructive heuristics for parallel machine scheduling with sequence-dependent setups and common servers. *Computers & Industrial Engineering*, 172:108586, 2022. doi:10.1016/j.cie.2022.108586.

32 Keld Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. doi:10.1016/S0377-2217(99)00284-2.

33 Yi-Feng Hung and Wei-Chih Chen. A heterogeneous cooperative parallel search of branch-and-bound method and tabu search algorithm. *Journal of Global Optimization*, 51(1):133–148, September 2011. doi:10.1007/s10898-010-9626-5.

34 IBM Corporation. IBM ILOG CPLEX CP Optimizer. https://www.ibm.com/products/ilog-cplex-optimization-studio, 2026. Accessed: 2026-03-02.

35 S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

36 J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 – International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995. doi:10.1109/ICNN.1995.488968.

37 Madjid Khichane, Patrick Albert, and Christine Solnon. Strong combination of ant colony optimization with constraint programming optimization. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 232–245, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-13520-0_26.

38 Philip Kilby, Patrick Prosser, and Paul Shaw. *Guided Local Search for the Vehicle Routing Problem with Time Windows*, pages 473–486. Springer US, Boston, MA, 1999. doi:10.1007/978-1-4615-5775-3_32.

39 S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi:10.1126/science.220.4598.671.

40 Philippe Laborie. IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In Willem-Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 148–162, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-01929-6_12.

41 Philippe Laborie and Daniel Godard. Self-adapting large neighborhood search: Application to single-mode scheduling problems. *Proceedings MISTA-07, Paris*, 8:276–284, 2007.

42 Leon Lan and Joost Berkhout. Pyjobshop: Solving scheduling problems with constraint programming in python, 2025. arXiv:2502.13483.

43 Helena Ramalhinho Lourenço. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83(2):347–364, 1995. EURO Summer Institute Combinatorial Optimization. doi:10.1016/0377-2217(95)00012-F.

44 Alan S. Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960. URL: http://www.jstor.org/stable/167204.

45 Bernd Meyer. *Hybrids of Constructive Metaheuristics and Constraint Programming: A Case Study with ACO*, pages 151–183. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-78295-7_6.

46 Laurent Michel, Andrew See, and Pascal Van Hentenryck. Transparent parallelization of constraint programming. *INFORMS J. on Computing*, 21(3):363–382, 2009. doi:10.1287/ijoc.1080.0313.

47 Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P Report 826, California Institute of Technology, Concurrent Computation Program, 1989.

48 Bahman Naderi, Rubén Ruiz, and Vahid Roshanaei. Mixed-integer programming vs. constraint programming for shop scheduling problems: New results and outlook. *INFORMS Journal on Computing*, 35(4):817–843, 2023. doi:10.1287/ijoc.2022.0179.

49 Eoin O'Mahony, Emmanuel Hébrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science (AICS'08)*, pages 210–216, Dublin, Ireland, 2008. URL: https://homepages.laas.fr/ehebrard/papers/aics2008.pdf.

50 Robert Pellerin, Nathalie Perrier, and François Berthaut. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 280(2):395–416, 2020. doi:10.1016/j.ejor.2019.01.063.

51 Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, pages 468–481, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-30201-8_35.

52 Vilím Petr and Pons Diego Olivier Fernandez. OptalCP Benchmark Collection. URL: https://github.com/ScheduleOpt/optalcp-benchmarks.

53 Vilím Petr and Pons Diego Olivier Fernandez. OptalCP: Local Constraint Programming Solver. https://optalcp.com/install#requirements, 2024. Accessed: 2025-03-30.

54 F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, 2008. Part Special Issue: Search-based Software Engineering. doi:10.1016/j.cor.2007.02.014.

55 A. Alan B. Pritsker, Lawrence J. Waiters, and Philip M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1):93–108, 1969.

56 Jakob Puchinger and Günther R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In José Mira and José R. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, pages 41–53, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11499305_5.

57 Jean-Charles Régin and Arnaud Malapert. Parallel constraint programming. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 337–379. Springer, Cham, 2018. doi:10.1007/978-3-319-63516-3_9.

58 Jean-Charles Régin, Mohamed Rezgui, and Arnaud Malapert. Embarrassingly parallel search. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, pages 596–610, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-40627-0_45.

59 Gerhard Reinelt. TSPLIB—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991. doi:10.1287/IJOC.3.4.376.

60 Mauricio G. C. Resende and Celso C. Ribeiro. *Runtime distributions*, pages 113–146. Springer New York, New York, NY, 2016. doi:10.1007/978-1-4939-6530-4_6.

61 J. B. Robinson. *On the Hamiltonian game (a traveling-salesman problem)*. RAND Corporation, Santa Monica, CA, 1949.

62 Mohammad Rohaninejad, Behdin Vahedi-Nouri, Zdeněk Hanzálek, and Reza Tavakkoli-Moghaddam and. An integrated lot-sizing and scheduling problem in a reconfigurable manufacturing system under workforce constraints. *International Journal of Production Research*, 62(11):3994–4013, 2024. doi:10.1080/00207543.2023.2253311.

63 T.P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000. doi:10.1109/4235.873238.

64 Mohammad Saidi-Mehrabad and Parviz Fattahi. Flexible job shop scheduling with tabu search algorithms. *The International Journal of Advanced Manufacturing Technology*, 32(5):563–570, March 2007. doi:10.1007/s00170-005-0375-4.

65 Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. doi:10.1007/3-540-49481-2_30.

66 Paul Shaw, Bruno De Backer, and Vincent Furnon. Improved local search for CP toolkits. *Annals of Operations Research*, 115(1):31–50, September 2002. doi:10.1023/A:1021188818613.

67 Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987. doi:10.1287/opre.35.2.254.

68 Robert H. Storer, S. David Wu, and Renzo Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992. URL: http://www.jstor.org/stable/2632676.

69 E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993. Project Management anf Scheduling. doi:10.1016/0377-2217(93)90182-M.

70 El-Ghazali Talbi. Combining metaheuristics with mathematical programming, constraint programming and machine learning. *Annals of Operations Research*, 240(1):171–215, May 2016. doi:10.1007/s10479-015-2034-y.

71 Sarosh Talukdar, Lars Baerentzen, Andrew Gove, and Pedro De Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4(4):295–321, December 1998. doi:10.1023/A:1009669824615.

72 Eduardo Uchoa, Diego Pecin, and Artur Pessoa. VRP Web – Vehicle Routing Problem Repository, 2025. Accessed: 2025-03-30. URL: http://vrp.galgos.inf.puc-rio.br/index.php/en/.

73 Petr Vilím, Philippe Laborie, and Paul Shaw. Failure-directed search for constraint-based scheduling. In *Integration of AI and OR Techniques in Constraint Programming: 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings 12*, pages 437–453. Springer, 2015. doi:10.1007/978-3-319-18008-3_30.

74 Weijun Xia and Zhiming Wu. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, 48(2):409–425, 2005. doi:10.1016/j.cie.2005.01.018.

75 Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop problems. In *Parallel Problem Solving from Nature 2*, volume 2, pages 283–292, January 1992.

76 Takeshi Yamada and Ryohei Nakano. *Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search*, pages 237–248. Springer US, Boston, MA, 1996. doi:10.1007/978-1-4613-1361-8_15.