https://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING

Vol. 00, No. 0, Xxxxx 0000, pp. 000–000

Submitted to *INFORMS Journal on Computing*

ISSN 0899-1499, EISSN 1526-5528

Domain-Independent Dynamic Programming and Constraint Programming Approaches for Assembly Line Balancing Problems with Setups

Jiachen Zhang
Department of Mechanical and Industrial Engineering, University of Toronto, jasonzjc@mie.utoronto.ca

J. Christopher Beck
Department of Mechanical and Industrial Engineering, University of Toronto, jcb@mie.utoronto.ca

Abstract. We propose domain-independent dynamic programming (DIDP) and constraint programming (CP) models to exactly solve type-1 and type-2 assembly line balancing problem with sequence-dependent setup times (SUALBP). The goal is to assign tasks to assembly stations and to sequence these tasks within each station, while satisfying precedence relations specified between a subset of task pairs. Each task has a given processing time and a setup time dependent on the previous task on the station to which the task is assigned. The sum of the processing and setup times of tasks assigned to each station constitute the station time and the maximum station time is called the cycle time. For type-1 SUALBP, the objective is to minimize the number of stations, given a maximum cycle time. For type-2 SUALBP, the objective is to minimize the cycle time, given the number of stations. On a set of diverse SUALBP instances, experimental results show that our approaches significantly outperform the state-of-the-art mixed integer programming models for SUALBP-1. For SUALBP-2, the DIDP model outperforms the state-of-the-art exact approach based on logic-based Benders decomposition. By closing 76 open instances for SUALBP-2, our results demonstrate the promise of DIDP for solving complex planning and scheduling problems.

Funding: This work is supported by the Natural Sciences and Engineering Research Council of Canada.

Key words: Domain-independent dynamic programming, Constraint programming, Assembly line balancing

1. Introduction

The *simple assembly line balancing problem* (SALBP) is a well-known production planning problem (Becker and Scholl 2006), with many applications in the production of automotive and household items (Boysen et al. 2022). The SALBP is often the last step of production, performing the final assembly of a product from previously manufactured parts. Commonly utilized for the production of standardized products, an assembly line is a series of interconnected workstations linked by a transportation system, such as a conveyor belt, where work pieces move through the stations in a predetermined order. At each station, a set of tasks is performed sequentially. Due to the technical structure of the products, the set of tasks must adhere to a partial precedence ordering: if (a, b) represents a precedence relation between task a and b, then b must be assigned to the same or a later station than a. If assigned to the same station, b must be after a in the task sequence. In a feasible production plan, each task is assigned to a single station, ensuring all precedence constraints are met. The tasks at each station form the station's workload, and the total time for these tasks is the station time. The cycle time is the maximum station time across all stations.

In many real production lines, setup operations such as tool changes, curing, or cooling processes are required between consecutive tasks (Kumar and Mahto 2013). The *assembly line balancing problem with setups* (SUALBP) extends the SALBP by integrating setup times into task scheduling (Andres et al. 2008). Specifically, tasks assigned to the same station must satisfy sequence-dependent setup times in the schedule. Scholl et al. (2013) further extended this problem by distinguishing between forward and backward setup times. A forward setup occurs when one task immediately follows another within the same cycle, while a backward setup must happen between the last task of one cycle and the first task of the next cycle on the same station (Zohali et al. 2022).

In the literature, SUALBP is also known as the *general assembly line balancing problem with setups* (Martino and Pastor 2010) and the *setup assembly line balancing and scheduling problem* (Scholl et al. 2013). SUALBP is categorized into two main types: SUALBP-1 minimizes the number of stations given a fixed cycle time and SUALBP-2 minimizes the cycle time given a fixed number of stations. Due to different application scenarios, other variants also exist in the literature (Wee and Magazine 1982, Becker and Scholl 2006).

There has been a variety of approaches applied to the SUALBP-1 variants including *mixed-integer programming* (MIP) (Akpinar and Baykasoğlu 2014a), constraint programming (Güner et al. 2023), hybrid bee colony algorithms (Akpinar and Baykasoğlu 2014b), Benders decomposition (Akpinar et al. 2017), simulated annealing (Özcan 2019), and variable neighborhood search (Yang and Cheng 2020). For SUALBP-2 variants, MIP models (Tang et al. 2016), genetic algorithms combined with dynamic programming (Yolmeh and Kianfar 2012), and meta-heuristics (Şahin and Kellegöz 2017) have been developed.

Narrowing our scope to standard SUALBP-1 and SUALBP-2, Andres et al. (2008) proposed a mathematical model and multiple heuristics for the former while Martino and Pastor (2010) developed heuristic algorithms. Later, Scholl et al. (2013) presented a new MIP model and multiple heuristics. For SUALBP-2, Seyed-Alagheband et al. (2011) introduced a MIP model based on the formulation of Andrés et al. (2008) and devised a simulated annealing meta-heuristic algorithm. Esmaeilbeigi et al. (2016) proposed improved MIP models for both SUALBP-1 and SUALBP-2 that represent the state-of-the-art exact techniques for the former. More recently, Zohali et al. (2022) developed the state-of-the-art exact approach for SUALBP-2 using logic-based Benders decomposition.

In this work, we create novel *domain-independent dynamic programming* (DIDP) and *constraint programming* (CP) models for both types of SUALBP. The superior performance of the proposed DIDP models against the state-of-the-art approaches demonstrates the promise of this emerging exact method for solving complex planning and scheduling problems.

Main contributions. Our contributions are four novel optimization models (DIDP and CP) for SUALBP-1 and SUALBP-2, and new state-of-the-art results of the proposed DIDP models compared to the best exact algorithms in the literature. We successfully close 76 open instances for SUALBP-2 and detect potential flaws in empirical results of Zohali et al. (2022). We also investigate a local improvement algorithm for the DIDP model of SUALBP-2.

Outline of the paper. In Section 2, we define the problem, summarize the notation, present the state-of-the-art MIP models and exact algorithms, and introduce domain-independent dynamic programming and constraint programming. The proposed CP models for SUALBP-1 and SUALBP-2 are presented in Section 3. Similarly, the DIDP models for SUALBP-1 and SUALBP-2 are presented in Section 4. A local improvement algorithm to solve the DIDP model of SUALBP-2 is briefly introduced in Section 5. In Section 6, experimental results on benchmark instances are presented, followed by discussions regarding potential flaws in previous work. Finally, we conclude this work in Section 7.

2. Background

In this section, we define the assembly line balancing problem with setups, summarize the notation, present the state-of-the-art MIP models, briefly introduce the state-of-the-art logic-based Benders decomposition for SUALBP-2, and review domain-independent dynamic programming and constraint programming methodologies.

Table 1 Notation and definition for SUALBP-1 and SUALBP-2 (Esmaeilbeigi et al. 2016).

Notation	Definition
n	the number of tasks
V	the set of tasks, i.e., $V = \{1, 2, ..., n\}$
i, j, v	the task indices
$\mathcal{E}$	the set of precedence relations, i.e., $(i, j) \in \mathcal{E}$ if $i \in V$ precedes j
t_i	the execution time of task $i \in V$
$P_i(P_i^*)$	the set of direct (all) predecessors of task $i \in V$
$F_i(F_i^*)$	the set of direct (all) successors of task $i \in V$
$\overline{c}(\underline{c})$	the upper (lower) limit of the cycle time
$\overline{m}(\underline{m})$	the upper (lower) limit of the station number
E_i	the earliest assignable station for task $i \in V$, e.g., $E_i = \lceil \frac{t_i + \sum_{j \in P_i^*} t_j}{\overline{c}} \rceil$
L_i	the latest assignable station for task $i \in V$, e.g., $L_i = \overline{m} + 1 - \lceil \frac{t_i + \sum_{j \in F_i^*} t_j}{\overline{c}} \rceil$
$KD(KP)$	the set of definite (possible) stations, i.e., $KD = \{1, ..., \underline{m}\}$, and $KP = \{\underline{m} + 1, ..., \overline{m}\}$
K	the set of stations, i.e., $K = KD \cup KP$
k	the station index
FS_i	the set of assignable stations for task $i \in V$, i.e., $FS_i = \{E_i, E_i + 1, ..., L_i\}$
FT_k	the set of tasks which can be assigned to station $k \in K$, i.e., $FT_k = \{i \in V \vert k \in FS_i\}$
A_i	the set of tasks that cannot be assigned to the station to which task $i \in V$ is assigned, e.g., $A_i = \{j \in V \vert FS_j \cap FS_i = \emptyset\}$
$F_i^F(P_i^F)$	the set of tasks which can directly follow (precede) task $i \in V$ in forward direction, i.e., $F_i^F = \{j \in V - (F_i^* - F_i) - P_i^* - A_i - \{i\}\}$ and $P_i^F = \{j \in V \vert i \in F_j^F\}$
$F_i^B(P_i^B)$	the set of tasks which can directly follow (precede) task $i \in V$ in backward direction, i.e., $F_i^B = \{j \in V - F_i^* - A_i\}$ and $P_i^B = \{j \in V \vert i \in F_j^B\}$
τ_{ij}	the forward setup times from task $i \in V$ to task $j \in F_i^F$
μ_{ij}	the backward setup times from task $i \in V$ to task $j \in F_i^B$
$\underline{\tau}_i$	the smallest forward setup time from any other task to task $i \in V$
$\underline{\mu}_i$	the smallest backward setup time from any other task to task $i \in V$

2.1. Problem Definition and Notation

Following Esmaeilbeigi et al. (2016), SUALBP consists of n assembly tasks with precedence constraints, that require processing on m different, ordered assembly stations with cycle time c. In the two main types of SUALBP, one of m or c is the objective while the maximum value of the other is fixed. All stations are capable of performing all assembly tasks, and the tasks allocated to a specific station must be fully sequenced. If a task is assigned to station j, all of its successors must be assigned to the same or subsequent stations (i.e., $j, j+1, ..., m$). Tasks assigned to the same station must be sequenced to adhere to any precedence constraints. The processing time for each task is predetermined and deterministic. However, the setup time required before performing a task is dependent on the preceding task in the station's processing sequence.

For tasks assigned to a station, there are two types of sequence-dependent setups to consider: (i) a *forward setup*, which accounts for the setup times between any two consecutive tasks within the same cycle, and (ii) a *backward setup*, which accounts for the setup times between the last task of one cycle and the first task of the subsequent cycle on the same station. The sum of processing and the forward and backward setup times on a station constitutes the station time. The cycle time, c, is the maximum station time. The setups are not symmetric, i.e., the setup time from task i to j might be different from the setup time from task j to i. Nevertheless, the forward setups satisfy the triangle inequality.

In SUALBP-1, the cycle time c is fixed, and the goal is to minimize the number of stations m. Conversely, in SUALBP-2, the number of stations m is fixed, and the objective is to minimize the cycle time c. In both scenarios, the key decisions involve (i) assigning tasks to stations and (ii) determining the sequence of tasks at each station.

We use the notation proposed by Esmaeilbeigi et al. (2016), as shown in Table 1 for both SUALBP-1 and SUALBP-2. For SUALBP-1, $\overline{c} = \underline{c} = c$. For SUALBP-2, $\overline{m} = \underline{m} = m$. To obtain all the parameters in the table, we adapt the preprocessing techniques in the literature (Kuroiwa and Beck 2023a, Esmaeilbeigi et al. 2016, Zohali et al. 2022).

2.2. State-of-the-art MIP Model for SUALBP-1

The state-of-the-art MIP model is the *second station-based formulation* (SSBF) proposed by Esmaeilbeigi et al. (2016). Since the SSBF model can be adapted to both SUALBP-1 and SUALBP-2, we call it SSBF-1 in this section. The decision variables are:

- x_{ik}: binary variable, $x_{ik} = 1$ iff task $i \in V$ is assigned to station $k \in FS_i$.
- z_i: integer variable representing the index of the station task $i \in V$ is assigned to.

- u_k: binary variable, $u_k = 1$ iff any task is assigned to station k.
- g_{ijk}: binary variable, $g_{ijk} = 1$ iff task i is performed immediately before task j on station k.
- h_{ijk}: binary variable, $h_{ijk} = 1$ iff task i is the last and j is the first task on station k.
- r_i: integer variable encoding the rank of task i in a sequence of all tasks. The sequence is composed of the task sequences on all the active stations, i.e., starting from the first station and ending at the last active station.

The SSBF-1 MIP model proposed by Esmaeilbeigi et al. (2016) is as follows.

$$\min \sum_{k \in KP} u_k + \underline{m} \tag{1a}$$

$$\text{s.t.} \sum_{k \in FS_i} x_{ik} = 1, \quad \forall i \in V, \tag{1b}$$

$$\sum_{k \in FS_i} k \cdot x_{ik} = z_i, \quad \forall i \in V, \tag{1c}$$

$$\sum_{i \in FT_k \cap F_i^F} g_{ijk} + \sum_{i \in FT_k \cap F_i^B} h_{ijk} = x_{ik}, \quad \forall i \in V, \forall k \in FS_i, \tag{1d}$$

$$\sum_{i \in FT_k \cap P_j^F} g_{ijk} + \sum_{i \in FT_k \cap P_j^B} h_{ijk} = x_{jk}, \quad \forall j \in V, \forall k \in FS_j, \tag{1e}$$

$$\sum_{i \in FT_k} \sum_{j \in (FT_k \cap F_i^B)} h_{ijk} = 1, \quad \forall k \in KD, \tag{1f}$$

$$\sum_{i \in FT_k} \sum_{j \in (FT_k \cap F_i^B)} h_{ijk} = u_k, \quad \forall k \in KP, \tag{1g}$$

$$r_i + 1 + (n - |F_i^*| - |P_j^*|) \cdot (\sum_{k \in (FS_i \cap FS_j)} g_{ijk} - 1) \le r_j, \quad \forall i \in V, \forall j \in F_i^F, \tag{1h}$$

$$r_i + 1 \le r_j, \quad \forall (i,j) \in \mathcal{E}, \tag{1i}$$

$$z_i \le z_j, \quad \forall (i,j) \in \mathcal{E}, \tag{1j}$$

$$\sum_{i \in FT_k} t_i \cdot x_{ik} + \sum_{i \in FT_k} \sum_{j \in (FT_k \cap F_i^F)} \tau_{ij} \cdot g_{ijk} + \sum_{i \in FT_k \cap P_i^B} \mu_{ij} \cdot h_{ijk} \le \overline{c}, \quad \forall k \in KD, \tag{1k}$$

$$\sum_{i \in FT_k} t_i \cdot x_{ik} + \sum_{i \in FT_k} \sum_{j \in (FT_k \cap F_i^F)} \tau_{ij} \cdot g_{ijk} + \sum_{i \in FT_k \cap P_i^B} \mu_{ij} \cdot h_{ijk} \le \overline{c} \cdot u_k, \quad \forall k \in KP, \tag{1l}$$

$$\sum_{i \in FT_k \backslash \{j\}} x_{ik} \le (n - \underline{m} + 1) \cdot (1 - h_{jjk}), \quad \forall k \in K, \forall j \in FT_k, \tag{1m}$$

$$u_{k+1} \le u_k, \quad \forall k \in KP \backslash \{\overline{m}\}. \tag{1n}$$

$$g_{ijk} \in \{0,1\}, \quad \forall k \in K, \forall i \in FT_k, \forall j \in (FT_k \cap F_i^F), \tag{1o}$$

$$h_{ijk} \in \{0,1\}, \quad \forall k \in K, \forall i \in FT_k, \forall j \in (FT_k \cap F_i^B), \tag{1p}$$

$$|P_i^*| + 1 \le r_i \le n - |F_i^*|, \quad \forall i \in V, \tag{1q}$$

$$x_{ik} \in \{0,1\}, \quad \forall i \in V, \forall k \in FS_i, \tag{1r}$$

$$r_i, z_i \in \mathbb{Z}^+, \quad \forall i \in V, \tag{1s}$$

In SSBF-1, the objective function (1a) minimizes the number of stations. Constraint (1b) assigns each task to a station, while (1c) links variables x_{ik} and z_i. Constraints (1d) and (1e) ensure that each task at station k is part of a cycle, preceded and followed by exactly one other task. Constraints (1f) and (1g) enforce that only one backward setup occurs per cycle. Precedence relations within the same station are expressed by constraints (1h) and (1i), with (1j) extending this to tasks across different stations. Knapsack constraints (1k) and (1l) limit station times to the cycle time. Constraint (1m) restricts the allocation of tasks based on setup conditions. Constraint (1n) ensures stations are sequentially utilized, avoiding gaps. Finally, constraints (1o) to (1s) define variable domains.

Note that the decision variables r_i and z_i are set to continuous in Esmaeilbeigi et al. (2016). However, doing so results in infeasible solutions being labeled as feasible for some problem instances. In addition to the MIP model, Esmaeilbeigi et al. (2016) developed pre-processing techniques to prune the number of variables and constraints. We implement all these techniques but omit the details here.

2.3. State-of-the-art MIP Model for SUALBP-2

The decision variables of the state-of-the-art SSBF-2 MIP model proposed by Zohali et al. (2022) are:

- x_{ik}, z_i, g_{ijk}, h_{ijk}, r_i as defined in Section 2.2.
- c: continuous variable to represent the cycle time.

The SSBF-2 MIP model proposed by Zohali et al. (2022) is as follows.

$$\min c \tag{2a}$$

$$\text{s.t.} \quad (1b)-(1e), (1h)-(1j), (1o)-(1s), \tag{2b}$$

$$\sum_{i \in FT_k} \sum_{j \in (FT_k \cap F_i^B)} h_{ijk} = 1, \quad \forall k \in K, \tag{2c}$$

$$\sum_{i \in FT_k} t_i \cdot x_{ik} + \sum_{i \in FT_k} \sum_{j \in (FT_k \cap F_i^F)} \tau_{ij} \cdot g_{ijk} + \sum_{i \in FT_k \cap P_i^B} \mu_{ij} \cdot h_{ijk} \le c, \quad \forall k \in K, \tag{2d}$$

$$\sum_{i \in FT_k \setminus \{j\}} x_{ik} \leq (n - \underline{m} + 1) \cdot (1 - h_{jjk}), \quad \forall k \in K, \forall j \in FT_k, \tag{2e}$$

$$\sum_{i \in FT_k} x_{ik} \geq 1, \quad \forall k \in K, \tag{2f}$$

$$c + \overline{c} \cdot (\sum_{k \in FS_j} k \cdot x_{jk} - \sum_{k \in FS_i} k \cdot x_{ik}) \geq D_{ij}, \quad \forall i \in V, j \in (F_i^F \setminus A_i), \tag{2g}$$

$$\underline{c} \leq c \leq \overline{c}. \tag{2h}$$

In SSBF-2, the objective function (2a) aims to minimize the cycle time. Constraint (2c) ensures that each cycle includes exactly one backward setup. The knapsack constraint (2d) limits the station time to the cycle time. Constraint (2e) dictates that only task j is assigned to station k when $h_{jjk} = 1$. Constraint (2f) introduces valid inequalities, ensuring that there is at least one task at each station, given that $n > m$ (Zohali et al. 2022). Constraint (2g) provides additional valid inequalities, where D_{ij} is a lower bound on the cycle time c if tasks i and j are scheduled at the same station (Esmaeilbeigi et al. 2016).

As above, we implement all of Zohali et al.'s pre-processing techniques but omit the details here.

2.4. State-of-the-art Exact Approaches for SUALBP-1 and SUALBP-2

The MIP model in Section 2.2 is the state-of-the-art exact approach for SUALBP-1 (Esmaeilbeigi et al. 2016).

For the SUALBP-2 problem, the state-of-the-art exact method is the *full-featured logic-based Benders decomposition* (FFLBBD) (Zohali et al. 2022). This approach decomposes SUALBP-2 into a master problem and a series of subproblems, each corresponding to a station. The master problem focuses on task assignment to stations, considering only processing times and precedence constraints. Each subproblem sequences the tasks assigned to a station, accounting for sequence-dependent setup times. Optimal station schedules generate cuts that are fed back into the master problem, and a relaxation of the subproblems is included in the master to tighten the gap between the two levels. The LBBD algorithm also integrates pre-processing, relaxations, valid inequalities, and bounds. Both the master problem and subproblems are modeled as MIP formulations. Due to the complexity of this approach, we direct readers to the original paper (Zohali et al. 2022) for full details.

2.5. Constraint Programming

Constraint programming (CP) is an approach to combinatorial optimization originating in the artificial intelligence community (Hooker and van Hoeve 2018). CP features rich variable types (e.g., interval variables and graph variables (Dooms et al. 2005)) and global constraints that represent common combinatorial sub-structure and that have an associated inference algorithm (Rossi et al. 2006). CP allows greater flexibility and extensibility in the types of constraints (Laborie et al. 2018) compared to MIP, resulting in the development of many global constraints that capture modeling and inference techniques. However, as CP relies on local inference algorithms within global constraints to limit search and propagation of information between constraints via variable domains, a global problem view is often lacking, frequently resulting in a relatively loose CP dual bound of the global objective (Hooker 2002, Bockmayr and Kasper 1998, Achterberg 2009). Nonetheless, Kuroiwa and Beck (2023b) showed that CP is able outperform MIP on finding and proving optimal solutions for five out of nine problem classes tested and is often able to find better quality solutions than MIP across different time limits.

CP has been used to solve SALBP-1 and SALBP-2 and outperforms MIP for larger instances (Bukchin and Raviv 2018). Recently, CP has been applied to the two-sided disassembly line balancing problem with AND/OR precedence and sequence-dependent setup times, where each station is composed of two independent workstations (sides) and precedence constraints must be addressed considering the predecessors of a task on both sides of the assembly line (Çil et al. 2022, Kizilay 2022). CP has also been used in multi-manned SUALBP-1 to minimize the number of workers required for assigned tasks (Güner et al. 2023). In all cases, CP outperforms MIP for these non-standard variants. This paper proposes CP models inspired by Çil et al. (2022).

2.6. Domain-Independent Dynamic Programming

The *Domain-Independent Dynamic Programming* (DIDP) framework is a recent exact method for combinatorial optimization problems (Kuroiwa and Beck 2023a). Applied to problems such as traveling salesman with time windows, multi-commodity pickup and delivery traveling salesman, and SALBP-1 (Kuroiwa and Beck 2023b), DIDP outperformed MIP and CP models in six out of nine tested problem classes. This success motivates us to investigate using DIDP to solve SUALBP.

DIDP is a model-and-solve framework for dynamic programming. A problem is represented as a dynamic programming model expressed in a domain-independent modeling language, *Dynamic Programming Description Language* (DyPDL). As in MIP and CP, the model is then solved by a generic solver.

A DyPDL model is expressed by a tuple $\langle \mathcal{V}, \mathcal{S}^0, \mathcal{K}, \mathcal{T}, \mathcal{B}, \mathcal{C}, h \rangle$, where $\mathcal{V} = \{v_1, ..., v_n\}$ is the set of state variables with $\mathcal{S}^0$ being the target state and $\mathcal{B}$ the set of base cases. $\mathcal{T}$ is the set of transitions between states, while $\mathcal{C}$ is the set of state constraint. $\mathcal{K}$ is the set of constants and h is the dual bound. A solution to a DIDP model can be found by solving the recursion to determine the optimal cost of S^0. The recursion is a sequence of transitions from S^0 to a state satisfying one of the base cases.

Existing solvers for DIDP are based on heuristic state-space search approaches that have been developed in the field of artificial intelligence over the past 50 years (Hart et al. 1968). In particular, Kuroiwa and Beck (2023b) showed that *complete anytime beam search* (CABS) (Zhang 1998) achieves the best performance compared to other state-space search variants on each of the nine combinatorial optimization problems that were used for evaluation. Further, CABS outperformed both MIP and CP solvers (solving MIP and CP models, respectively) on six of the nine problem classes. As the name suggests, CABS, is a complete search approach: given enough time will find optimal solutions and prove optimality (Zhang 1998). For a more detailed description of CABS, please see our online appendix (Zhang and Beck 2024a).

DIDP has been applied to SALBP-1 (Kuroiwa and Beck 2023a), where the DIDP formulation is inspired by the state-of-the-art branch-bound-and-remember approach (Morrison et al. 2014). Our novel DIDP formulations for SUALBP are inspired by this DIDP model.

3. CP Models for SUALBP-1 and SUALBP-2

In this section, we present the novel CP models for SUALBP-1 and SUALBP-2. We summarize the additional notation used in the CP models in Table 2.

3.1. CP model for SUALBP-1

The decision variables of the proposed CP model are:

- $task_i$, $\overline{task}_i^k$, d_s^k, d_t^k, $d_s^{k'}$, and $d_t^{k'}$ are defined as in Table 2.
- u_k: binary variable, $u_k = 1$ iff any task is assigned to station k.

Table 2 Summary of notation in the CP models.

Symbol	Definition
$task_i$ ($\overline{task}_i^k$)	(optional) interval variable (Laborie et al. 2018) of task i (on station k $k \in FS_i$), with a size of t_i and a range between 0 and c
d_s^k (d_t^k)	interval variables of dummy first (last) task on station k, with size 0
$d_s^{k'}$	the next task after the dummy first task on station k
$d_t^{k'}$	the previous task before the dummy last task on station k

The CP model is as follows:

$$\min \sum_{k\in KP} u_k + \underline{m} \tag{3a}$$

$$\text{s.t. } \texttt{Alternative}(task_i, \overline{task}_i^{E_i}, ..., \overline{task}_i^{L_i}), \quad \forall i \in V, \tag{3b}$$

$$\texttt{NoOverlap}(\{\overline{task}_i^k \cup \{d_s^k, d_t^k\}, \forall i \in FT_k\}, \{\tau_{ij}, \forall i, j \in FT_k\}), \quad \forall k \in KD \cup KP, \tag{3c}$$

$$\texttt{First}(\{\overline{task}_i^k, \forall i \in FT_k\} \cup \{d_s^k, d_t^k\}, d_s^k), \quad \forall k \in KD \cup KP, \tag{3d}$$

$$\texttt{Last}(\{\overline{task}_i^k, \forall i \in FT_k\} \cup \{d_s^k, d_t^k\}, d_t^k), \quad \forall k \in KD \cup KP, \tag{3e}$$

$$\texttt{PresenceOf}(\overline{task}_i^k) \le 1, \quad \forall k \in KD, \forall i \in FT_k, \tag{3f}$$

$$\texttt{PresenceOf}(\overline{task}_i^k) \le u_k, \quad \forall k \in KP, \forall i \in FT_k, \tag{3g}$$

$$\texttt{TypeOfNext}(d_s^k) = d_s^{k'}, \quad \forall k \in KD \cup KP, \tag{3h}$$

$$\texttt{TypeOfPrev}(d_t^k) = d_t^{k'}, \quad \forall k \in KD \cup KP, \tag{3i}$$

$$\texttt{StartOf}(d_t^k) + \texttt{Element}(\mu_{d_t^{k'} d_s^{k'}}) \le c, \quad \forall k \in KD, \tag{3j}$$

$$\texttt{StartOf}(d_t^k) + \texttt{Element}(\mu_{d_t^{k'} d_s^{k'}}) \le c \cdot u_k, \quad \forall k \in KP, \tag{3k}$$

$$\texttt{EndBeforeStart}(\overline{task}_i^k, \overline{task}_j^k), \quad \forall k \in KD \cup KP, \forall i \in FT_k, \forall j \in FT_k \cap F_i^*, \tag{3l}$$

$$\sum_{k\in FS_i} k \cdot \texttt{PresenceOf}(\overline{task}_i^k) \le \sum_{k\in FS_j} k \cdot \texttt{PresenceOf}(\overline{task}_j^k), \quad \forall i \in V, \forall j \in F_i^*, \tag{3m}$$

$$0 \le task_i \le c, \quad \forall i \in V, \tag{3n}$$

$$0 \le \overline{task}_i^k \le c, \quad \forall i \in V, \forall k \in FS_i, \tag{3o}$$

$$0 \le d_s^k, d_t^k \le c, \quad \forall k \in FS_i, \tag{3p}$$

$$u_k = 1, \quad \forall k \in KD, \tag{3q}$$

$$u_k \in \{0,1\}, \quad \forall k \in KP. \tag{3r}$$

The objective (3a) minimizes the number of stations used. Constraint (3b) aligns fixed and optional interval variables. Constraint (3c) ensures interval variables form a sequence on each station. Constraints (3d) and (3e) require d_s^k and d_t^k to be the first and last tasks on station k. Constraints (3f) and (3g) restrict task assignments to open stations. Constraints (3h) and (3i) ensure that $d_s^{k'}$ follows d_s^k and $d_t^{k'}$ precedes d_t^k. Constraints (3j) and (3k) account for backward setup time using an `element` constraint. Constraint (3l) ensures that within the same station, a follower task starts after its predecessor. Constraint (3m) prevents a task from being assigned to a later station than its followers. Constraints (3n) to (3r) define variable domains.

3.2. CP model for SUALBP-2

The decision variables of the proposed CP model are:

- $task_i$, $\overline{task}_i^k$, d_s^k, d_t^k, $d_s^{k'}$, and $d_t^{k'}$ as defined in Table 2.
- c: integer variable to represent the cycle time.

The CP model is as follows:

$$\min c \tag{4a}$$

$$\text{s.t.} \quad (3b)-(3e),(3h),(3i),(3l),(3m), \tag{4b}$$

$$\texttt{PresenceOf}(\overline{task}_i^k) \leq 1, \quad \forall k \in K, \forall i \in FT_k, \tag{4c}$$

$$\texttt{StartOf}(d_t^k) + \texttt{Element}(\mu_{d_t^{k'} d_s^{k'}}) \leq c, \quad \forall k \in K, \tag{4d}$$

$$0 \leq task_i \leq \overline{c}, \quad \forall i \in V, \tag{4e}$$

$$0 \leq \overline{task}_i^k \leq \overline{c}, \quad \forall i \in V, \forall k \in FS_i, \tag{4f}$$

$$0 \leq d_s^k, d_t^k \leq \overline{c}, \quad \forall k \in FS_i, \tag{4g}$$

$$\underline{c} \leq c \leq \overline{c}. \tag{4h}$$

The objective (4a) minimizes cycle time. Constraint (4c) restricts task assignments to open stations. Constraint (4d) incorporates backward setup time using an `element` constraint. Constraints (4e) to (4h) define variable domains.

Some CP frameworks such as MiniZinc (Nethercote et al. 2007) do not support optional interval variables. A CP model for SUALBP in that context would need many integer variables to represent task assignment explicitly and also many interval variables to address the sequencing problems in stations, with some linking constraints for these two sets of variables. The propagation would likely be weak as inferences based on sequencing decisions are obtained much deeper in the search tree compared to when assignment decisions are made.

4. DIDP Models for SUALBP-1 and SUALBP-2

In this section, we present the novel DIDP models for SUALBP-1 and SUALBP-2. We summarize the addtional notation used in the DIDP models in Table 3.

4.1. DIDP model for SUALBP-1

A DIDP model can be defined using a state-transition system. A state in our DIDP model is defined by the set of unscheduled tasks, the current station that tasks are being assigned to, the first and previous (i.e., most recently assigned) tasks on the current station, and the remaining cycle time

Table 3 Summary of notation in the DIDP models.

Symbol	Definition
U	set variable for unscheduled tasks
κ	integer (resource) variable representing the index of the current station
d_s	dummy task with 0 processing time and 0 forward and backward setup
p (f)	element variable for the previous (first) task of the current station
r	integer resource variable for the remaining time of the current station
t^c	integer resource variable for the used time of the current station

on the current station. Transitions consist of assigning a task as the next one on the current station, closing the current station to allow no further tasks to be assigned, and assigning a task as the first task on a station.

In our DIDP model, we use `set` variables and `element` variables, where a `set` variable represents a group of elements such as customers, tasks, or items, and an `element` variable represents an element of a set. DyPDL allows the use of `resource` variables to capture dominance relations between states. In our DIDP model for SUALBP-1, we use an integer resource variable to represent the remaining time of the current station. If two states have the same variable values except for the resource variable, the state with a larger remaining time dominates and the other state can be soundly pruned. Similarly, in our DIDP model for SUALBP-2, we consider an integer resource variable representing the used time of the current station. Then the state with a smaller used-time value dominates another with all the other variable values being the same.

For the proposed DIDP model, we first present the state variables and the base cases of the model. We then present the recursive function in a dynamic programming form. An alternative perspective on this problem definition by defining the state transitions that implement the recursive function is provided in our online appendix (Zhang and Beck 2024a).

State variables.

- U: set variable for unscheduled tasks. In the target state (i.e., the initial state), $U = V$.
- κ: integer resource variable representing the index of the current station. In the target state, $\kappa = 0$. A smaller κ is better. The state variable κ is used only for computing a state-based dual bound.
- p: element variable for the previous task of the current station. In the target state, $p = d_s$ where d_s is a dummy task with 0 processing time and 0 forward and backward setup with any other tasks.

- f: element variable for the first task of the current station. In the target state, $f = d_s$. A state keeps track of this task in order to handle the backward setup time when closing a station.
- r: integer resource variable for the remaining time (cycle time minus used time) of the current station. In the target state, $r = 0$. A larger r is better.

Base case. A base case states a set of conditions that terminate the recursion. The base case of the DIDP model is: $U = \emptyset \wedge f = d_s$. Note that $f = d_s$ is necessary since the current station has to be closed to correctly incorporate the backward setup time.

Recursive function. We represent the cost of a state by $\mathcal{V}(U, \kappa, p, f, r)$ and define $U_1 = \{j \in U \mid U \cap P^*_j = \emptyset\}$ as the set of tasks with all predecessors scheduled. Then, $U_2 = \{j \in U_1 \mid r \geq t_j + \tau_{pi}\}$ is the subset of U_1 that can be assigned next to the current station considering processing and forward setup times. Finally, $U_3 = \{j \in U_1 \mid r \geq t_j + \tau_{pi} + \mu_{if}\}$ further considers backward setup times. The recursive function of the DIDP model is then defined as follows:

$$\texttt{compute}\ \mathcal{V}(V, 0, d_s, d_s, 0) \tag{5a}$$

$$\mathcal{V}(U, \kappa, p, f, r) = \tag{5b}$$

$$\begin{cases} 0 & \text{if } U = \emptyset, f = d_s, & \text{(i)} \\ 1 + \min_{j \in U_1} \mathcal{V}(U \backslash \{j\}, \kappa + 1, j, j, c - t_j) & \text{if } U_1 \neq \emptyset, f = d_s, & \text{(ii)} \\ \min_{j \in U_2} \mathcal{V}(U \backslash \{j\}, \kappa, j, f, r - t_j - \tau_{pj}) & \text{if } U_2 \neq \emptyset, f \neq d_s, & \text{(iii)} \\ \mathcal{V}(U, \kappa, d_s, d_s, 0) & \text{if } U_3 = \emptyset, f \neq d_s, \mu_{pf} \leq r, & \text{(iv)} \\ \infty & \text{otherwise,} & \text{(v)} \end{cases}$$

$$U_1 = \{j \in U | U \cap P^*_j = \emptyset\}, \quad U_2 = \{j \in U_1 | r \geq t_j + \tau_{pi}\}, \quad U_3 = \{j \in U_1 | r \geq t_j + \tau_{pi} + \mu_{if}\}$$

$$\mathcal{V}(U, \kappa, p, f, r) \leq \mathcal{V}(U, \kappa', p, f, r), \quad \text{if } \kappa \leq \kappa', \tag{5c}$$

$$\mathcal{V}(U, \kappa, p, f, r) \leq \mathcal{V}(U, \kappa, p, f, r'), \quad \text{if } r \geq r', \tag{5d}$$

$$\mathcal{V}(U, \kappa, p, f, r) \geq \max \begin{cases} \left\lceil \frac{\underline{\mu}_f + \sum_{i \in U} (\underline{\tau}_i + t_i) - (\overline{m} - \kappa) \cdot (\max_{i \in U} \underline{\tau}_i) + \max(\underline{m} - \kappa, 0) \cdot (\min_{i \in U} \underline{\mu}_i) - r}{c} \right\rceil, & \text{(i)} \\ \left\lceil \frac{\underline{\mu}_f + \sum_{i \in U} t_i - r}{c} \right\rceil, & \text{(ii)} \\ \sum_{i \in U} w_i^2 + \lceil \sum_{i \in U} w_i'^2 - l^2 \rceil, & \text{(iii)} \\ \lceil \sum_{i \in U} w_i^3 - l^3 \rceil, & \text{(iv)} \\ 0. & \text{(v)} \end{cases} \tag{5e}$$

The term (5a) computes the cost of the target state. Equation (5b) is the main recursion of the DIDP model. Specifically, (5b-i) refers to the base case, while (5b-ii) corresponds to assigning the

Table 4 Numeric constants for calculating a knapsack-based dual bound.

t_i	(0, c/2)	c/2	(c/2, c]	t_i	(0, c/3)	c/3	(c/3, c/2)	2c/3	(2c/3, c]
w_i^2	0	0	1	w_i^3	0	1/3	1/2	2/3	1
$w_i'^2$	0	1/2	0						

first task to the current station. The recursion here can be understood as assigning the cost of the current state as equal to one more than the state where j is the first task on station $\kappa + 1$. Case (5b-iii) represents assigning the next task after others have already been assigned to the current station. Since the number of used stations remains unchanged, the cost of the current state equals that of the state where j is scheduled next on station κ. Task j then becomes the new last task on station κ. Cases (5b-iv) and (5b-v) correspond to closing the current station and detecting dead-ends, respectively. Inequalities (5c) and (5d) formulate state domination in two scenarios: if other variables are equal a state with smaller κ or larger remaining time dominates. Term (5e) states the state-based dual bounds. Both (5e-i) and (5e-ii) are novel dual bounds whose validity we prove below. (5e-iii) and (5e-iv) were first used for SALBP-1 (Kuroiwa and Beck 2023a) and are also valid here. (5e-v) is a default 0 dual bound if no problem-specific bound is provided.

The numeric constants w^2, w'^2, w^3 are indexed by a task i and depend on the processing time t_i, as shown in Table 4. These values are obtained by ignoring precedence relations and were originally proposed by Scholl and Klein (1997).

4.1.1. Correctness of Novel Dual Bounds

For the proposed state-based dual bound (5e-i), $\underline{\tau}_i$ is the smallest forward setup time from any task to task i and $\underline{\mu}_i$ is the smallest backward setup time from any task to task i. The value $\max_{i \in U} \underline{\tau}_i$ is the largest minimum forward setup time to any unscheduled task. Similarly, $\min_{i \in U} \underline{\mu}_i$ is the smallest minimum backward setup time to any unscheduled task.

THEOREM 1. *Term (5e-i) is a valid lower bound of the number of additional stations to be used at the current state.*

Proof. In Fig. 1 solid rectangles are stations that have tasks assigned, while dashed rectangles are stations that have no tasks assigned yet. Let the number of remaining stations to be used be β, the number of stations that are already used be α, and the current station, κ. r contributes to the

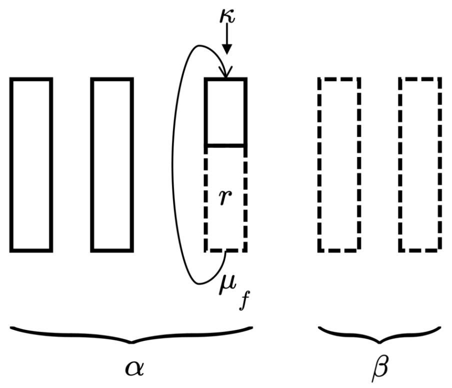

Figure 1 Illustration of dual bound (5e-i).

total time of the current station κ and is hence excluded from the calculation of β. Let the total time represented by the dashed rectangles in the best possible solution given the current state be T, then,

$$T = \underline{\mu}_f + \sum_{i \in U} t_i + \sum_{i \in U} \delta_i^* \tag{6}$$

where δ_i^* is either the forward setup time starting from task i (i.e., τ_{ij}, where j is the immediate successor of i on the assigned station) or the backward setup time (i.e., μ_{ij}, where j is the first and i is the last task on the assigned station) in the best possible solution reachable from the current state. Without knowing the best possible solution, the value of $\sum_{i \in U} \delta_i^*$ is unclear. A lower bound can be obtained as

$$\sum_{i \in U} \delta_i^* \geq \sum_{i \in U} \underline{\tau}_i - (\overline{m} - \kappa) \cdot \max_{i \in U} \underline{\tau}_i + \max(\underline{m} - \kappa, 0) \cdot \min_{i \in U} \underline{\mu}_i. \tag{7}$$

The inequality (7) is true since: (i) each station needs to consider a backward setup time and for the $\underline{m} - \kappa$ stations that are definitely used in the future, $(\underline{m} - \kappa) \cdot \min_{i \in U} \underline{\mu}_i$ is a lower bound on the real backward setup times on these stations,[1] however, since it is possible that $\underline{m} < \kappa$ at some states and negative backward setup times should not be added, $\max(\underline{m} - \kappa, 0) \cdot \min_{i \in U} \underline{\mu}_i$ is used; and (ii) as the remaining setup times are all forward, $\sum_{i \in U} \underline{\tau}_i - (\overline{m} - \kappa) \cdot \max_{i \in U} \underline{\tau}_i$ is a lower bound of the sum of all minimum forward setup times, minus all those that will be replaced by backward setup times in the $(\overline{m} - \kappa)$ possibly used stations. Thus,

$$T \geq \underline{\mu}_f + \sum_{i \in U} t_i + \sum_{i \in U} \underline{\tau}_i - (\overline{m} - \kappa) \cdot \max_{i \in U} \underline{\tau}_i + \max(\underline{m} - \kappa, 0) \cdot \min_{i \in U} \underline{\mu}_i. \tag{8}$$

[1] Recall that $\underline{m}$ is a lower bound on the number of machines uses.

Given that $\beta = \lceil \frac{T-r}{c} \rceil$, we finally have

$$\beta \geq \left\lceil \frac{\underline{\mu}_f + \sum_{i\in U}(\underline{\tau}_i + t_i) - (\overline{m} - \kappa)\cdot(\max_{i\in U}\underline{\tau}_i) + \max(\underline{m} - \kappa, 0)\cdot(\min_{i\in U}\underline{\mu}_i) - r}{c} \right\rceil. \tag{9}$$

A corner case is the target state where $\kappa = r = 0$ and $f = d_s$. In that case, the inequality is simplified to

$$\beta \geq \left\lceil \frac{\sum_{i\in U}(\underline{\tau}_i + t_i) + \underline{m}\cdot(\min_{i\in U}\underline{\mu}_i) - \overline{m}\cdot(\max_{i\in U}\underline{\tau}_i)}{c} \right\rceil, \tag{10}$$

which is also valid as $\sum_{i\in U}(\underline{\tau}_i + t_i) + \underline{m}\cdot(\min_{i\in U}\underline{\mu}_i) - \overline{m}\cdot(\max_{i\in U}\underline{\tau}_i)$ is smaller than the real total station time of all active stations. □

Similarly, if forward setup times are not considered and only the backward setup of the current station is counted, (5e-ii) is obtained. We omit the proof of its correctness. In most of cases, (5e-i) is greater than (5e-ii), but there are exceptions. We present the proof of the case where (5e-ii) is greater than or equal to (5e-i) as follows:

THEOREM 2. *Term (5e-i) does not dominate (5e-ii).*

Proof. Let $\overline{m} = \kappa + 2$ and $\kappa > \underline{m}$. Also, let $U = \{i, j\}$, $\underline{\tau}_i = 2$, and $\underline{\tau}_j = 12$. Then (5e-i) becomes less than or equal to (5e-ii) as follows:

$$\left\lceil \frac{\underline{\mu}_f + \sum_{i\in U} t_i + 2 + 12 - 2\times 12 + 0 - r}{c} \right\rceil = \left\lceil \frac{\underline{\mu}_f + \sum_{i\in U} t_i - 10 - r}{c} \right\rceil \leq \left\lceil \frac{\underline{\mu}_f + \sum_{i\in U} t_i - r}{c} \right\rceil.$$

Let $\underline{\mu}_f = 1, \sum_{i\in U} t_i = 50, r = 1$, and $c = 10$, then at this state, (5e-i)$= 4 < 5 =$(5e-ii). Thus, (5e-i) does not dominate (5e-ii).

4.2. DIDP model for SUALBP-2

This model is similar to the DIDP model of SUALBP-1, with a new state variable c to represent the cycle time and the replacement of r by t^c. In this model, we use t^c to keep track of the accumulated time on the current station. By contrast, in the model of SUALBP-1, we use r to keep track the remaining time of the current station. By definition $t^c + r = c'$ where c' is the cycle time of the current station. As above, the transition-centric view of the model definition is provided in our online appendix (Zhang and Beck 2024a).

State variables.

- U, p, f: these set variables are the same as the DIDP model for SUALBP-1.
- κ: integer variable for the current station. In the target state, $\kappa = 0$. Since the objective of SUALBP-2 is not to minimize the number of active stations, a smaller number of active stations is not preferred anymore, and so κ is not a resource variable.

- t^c: integer resource variable for the used time of the current station. In the target state, $t^c = 0$ represents that no task is assigned to the current station. A smaller t^c is better.
- c: integer resource variable for the cycle time. A smaller c is better. In the target state, $c = 0$. Note that the state variable c is used only for computing a state-based dual bound.

Base case. The base case of the DIDP model is: $U = \emptyset \wedge f = d_s$. Note that $f = d_s$ is necessary since in the base case one has to include the backward setup time for the last station.

Recursive function. We use $\mathcal{V}(U,\kappa,p,f,t^c,c)$ to represent the cost of a state. Let $U_1 = \{j \in U | U \cap P^*_j = \emptyset\}$. The recursive function of the DIDP model is as follows:

$$\texttt{compute}\ \mathcal{V}(V,0,d_s,d_s,0,0) \tag{11a}$$

$$\mathcal{V}(U,\kappa,p,f,t^c,c) = \tag{11b}$$

$$\begin{cases} 0 & \text{if } U=\emptyset, f=d_s, \quad \text{(i)} \\ \min_{j\in U_1}(\max(t_j, \mathcal{V}(U\backslash\{j\},\kappa+1,j,j,t_j,\max(c,t_j)))) & \text{if } U_1 \neq \emptyset, f = d_s, \kappa < m, \quad \text{(ii)} \\ \min \begin{cases} \min_{j\in U_1}(\max(t^c+t_j+\tau_{pj}, \mathcal{V}(U\backslash\{j\},\kappa,j,f, \\ \qquad t^c+t_j+\tau_{pj}, \max(c,t^c+t_j+\tau_{pj})))) & \text{if } U_1 \neq \emptyset, f \neq d_s, \quad \text{(iii)} \\ \max(t^c+\mu_{pf}, \\ \qquad \mathcal{V}(U,\kappa,d_s,d_s,t^c+\mu_{pf},\max(c,t^c+\mu_{pf}))) & \text{if } U_1 = \emptyset, f \neq d_s, \quad \text{(iv)} \end{cases} \\ \infty & \text{otherwise}, \quad \text{(v)} \end{cases}$$

$$U_1 = \{j \in U | U \cap P^*_j = \emptyset\},$$

$$\mathcal{V}(U,\kappa,p,f,t^c,c) \leq \mathcal{V}(U,\kappa,p,f,t^{c'},c), \quad \text{if } t^c \leq t^{c'}, \tag{11c}$$

$$\mathcal{V}(U,\kappa,p,f,t^c,c) \leq \mathcal{V}(U,\kappa,p,f,t^c,c'), \quad \text{if } c \leq c', \tag{11d}$$

$$\mathcal{V}(U,\kappa,p,f,t^c,c) \geq \max \begin{cases} \left\lceil \frac{\sum_{i\in U}(\underline{\tau}_i+t_i)+t^c+(m-\kappa)\cdot(\min_{i\in U}\underline{\mu}_i-\max_{i\in U}\underline{\tau}_i)+\underline{\mu}_f}{\min(m,m-\kappa+1)} - c \right\rceil, & \text{(i)} \\ \left\lceil \frac{\sum_{i\in U} t_i+t^c+\underline{\mu}_f}{\min(m,m-\kappa+1)} - c \right\rceil, & \text{(ii)} \\ 0. & \text{(iii)} \end{cases} \tag{11e}$$

The term (11a) is to compute the objective of the target state. Equation (11b) is the main recursion of the DIDP model. Specifically, (11b-i) handles the base cases, while (11b-ii) refers to assigning the first task to the current station. The recursion here can be understood as assigning the cost of the current state as equal to the maximum station time of the state where j is the first task on station $\kappa+1$. Case (11b-iii) corresponds to assigning the next task to a station after other tasks have

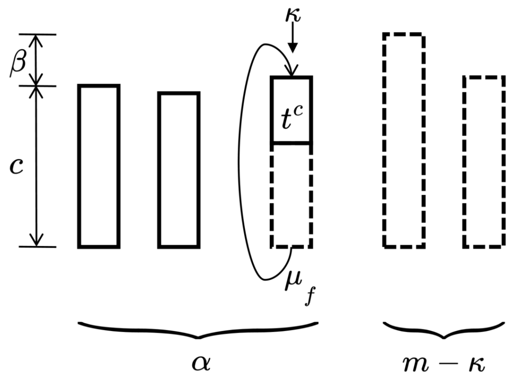

Figure 2 Illustration of dual bound (11e-i).

been assigned. Cases (11b-iv) and (11b-v) deal with closing the current station and dead-ends, respectively. Inequality (11c) and (11d) describe the state domination in two scenarios: (i) smaller used time dominates larger one and (ii) smaller cycle time dominates larger one, given all the other state variables are the same. Term (11e) formulates two novel state-based dual bounds and a default 0 dual bound.

4.2.1. Correctness of Novel Dual Bounds

THEOREM 3. *Term (11e-i) is a valid dual bound of the cycle time at the current state.*

Proof. In Fig. 2, solid rectangles are stations that have tasks assigned, while dashed rectangle are stations that have no tasks assigned yet. Let the number of stations that are already used be α, the current station be κ, the number of remaining stations to be used be $m-\kappa$, the current cycle time be c, and the difference between the future larger cycle time and c be β. Let the total time represented by the dashed rectangles in the best possible solution given the current state be T; t^c contributes to the total time of the current station κ and is hence excluded from the calculation of T. Then,

$$T = \underline{\mu}_f + \sum_{i\in U} t_i + \sum_{i\in U} \delta_i^*. \tag{12}$$

where δ_i^* is either the forward or the backward setup time ending at task i as defined in the proof of Theorem 1. Similar to the proof of Theorem 1, the following inequality is valid.

$$\sum_{i\in U} \delta_i^* \geq \sum_{i\in U} \underline{\tau}_i - (m-\kappa)\cdot \max_{i\in U} \underline{\tau}_i + (m-\kappa)\cdot \min_{i\in U} \underline{\mu}_i. \tag{13}$$

Thus,

$$T \geq \underline{\mu}_f + \sum_{i\in U} t_i + \sum_{i\in U} \underline{\tau}_i - (m-\kappa)\cdot \max_{i\in U} \underline{\tau}_i + (m-\kappa)\cdot \min_{i\in U} \underline{\mu}_i. \tag{14}$$

Given that $\beta = \lceil \frac{T+t^c}{m-\kappa+1} - c \rceil$, we finally have

$$\beta \geq \left\lceil \frac{\sum_{i \in U}(\underline{\tau}_i + t_i) + t^c + (m-\kappa)\cdot(\min_{i\in U}\underline{\mu}_i - \max_{i\in U}\underline{\tau}_i) + \underline{\mu}_f}{(m-\kappa+1)} - c \right\rceil. \tag{15}$$

Also, $\kappa \leq m$ is always true, as the only case that can increase κ requires $\kappa < m$, as shown in (11b-ii). Thus, $m - \kappa + 1 \geq 1$ and we do not need to handle 0 divisors in the dual bounds.

Another corner case is the target state where $\kappa = t^c = c = 0$ and $f = d_s$. In that case, the inequality is simplified to

$$\beta \geq \left\lceil \frac{\sum_{i \in U}(\underline{\tau}_i + t_i) + m\cdot(\min_{i\in U}\underline{\mu}_i - \max_{i\in U}\underline{\tau}_i)}{m} \right\rceil. \tag{16}$$

Therefore, we finally get

$$\beta \geq \left\lceil \frac{\sum_{i \in U}(\underline{\tau}_i + t_i) + t^c + (m-\kappa)\cdot(\min_{i\in U}\underline{\mu}_i - \max_{i\in U}\underline{\tau}_i) + \underline{\mu}_f}{\min(m, m-\kappa+1)} - c \right\rceil. \tag{17}$$

□

Similarly, (11e-ii) is obtained if forward and backward setup times are not considered and we omit the proof of its correctness. In majority of cases, (11e-i) leads to a larger dual bound than (11e-ii). Nevertheless, we prove the necessity of (11e-ii) by displaying a case where (11e-ii) needs to be strictly greater than (11e-i).

THEOREM 4. *Term (11e-i) does not dominate (11e-ii).*

Proof. Let $m = \kappa + 2$, $U = \{i, j\}$, $\underline{\tau}_i = 2$, and $\underline{\tau}_j = 12$. Also, let $\underline{\mu}_i = 1$, $\underline{\mu}_j = 1$, and $\underline{\mu}_f = 1$. Then (11e-i) becomes less than or equal to (11e-ii) as follows:

$$\left\lceil \frac{\sum_{i\in U} t_i + 2 + 12 + 2\times(1-12) + 1 + t^c}{\min(m, 2+1)} - c \right\rceil = \left\lceil \frac{\sum_{i\in U} t_i - 7 + t^c}{\min(m,3)} - c \right\rceil \leq \left\lceil \frac{\sum_{i\in U} t_i + t^c}{\min(m,3)} - c \right\rceil.$$

Let $\sum_{i\in U} t_i = 30, t^c = 1, m = 5$, and $c = 5$, then at this state, (11e-i)= $3 < 6$ =(11e-ii). Thus, (11e-i) does not dominate (11e-ii).

4.3. DIDP vs Decision Diagrams

Given dynamic programming models, Decision Diagram (DD) based optimization (Bergman et al. 2016) is a potential alternative to DIDP. Although there is no existing work that uses DD to solve SALBP or SUALBP, if a problem-specific merge operator is developed, it may be possible to solve the proposed models with DD-based approaches (Gillard et al. 2021). However, as there are currently no DD-based solvers that can read DyPDL models and the creation of a valid merge operator is a contribution in itself, we leave this investigation for future work.

Algorithm 1: Local Improvement

```
Input: $\mathbf{X}$ - incumbent solution of the original problem
1 $K, \{c_k, \forall k \in K\} \leftarrow$ StationTimeDecreasing($\mathbf{X}$);
2 repeat
3     $k \leftarrow$ pop($K$);
4     $\mathbf{X}'_k, c'_k \leftarrow$ LocalImprovement($\mathbf{X}_k$);
5 until $c'_k \geq c_k$ or $c'_k \geq c_{k+1}$;
6 $\mathbf{X}' \leftarrow$ UpdateSolution($\mathbf{X}, \mathbf{X}'_{\mathbf{k}}$);
7 return $\mathbf{X}', c'_k$;
```

5. Local Improvement for SUALBP-2

In this section, we present a local improvement algorithm based on the DIDP model for SUALBP-2. The idea is simple: given a feasible solution of SUALBP-2, the task schedule of each station can be independently re-optimized to reduce the station time, which might lead to a better cycle time. We call this process the "local improvement".

The local improvement method can be done combined with any anytime search algorithm. If the original algorithm is exact, it remains exact after the combination. Whenever a new incumbent with objective c is found for SUALBP-2, we try to refine this solution. Specifically, we first sort the stations in a decreasing order of station time. Then starting from the first station, we search for the sequence that minimizes station time. If no improvement exists on a station, the local improvement algorithm cannot (further) decrease the cycle time and it exits. If a better solution with objective c' is hence found, we return to where the SUALBP-2 algorithm paused with c' as the incumbent cost.

The *local improvement* (LI) algorithm is shown in Algorithm 1. The initialization of the algorithm is conducted in line 1, where the stations are sorted in a decreasing order of the station times in the incumbent solution $\mathbf{X}$. K is the set of sorted stations while c_k is the station time of station $k \in K$. In the loop from line 2 to line 5, each station in K is popped in order and the corresponding station time is re-optimized. If the station time is not reduced or the reduced station time is greater than or equal to the station time of the next station in K, the local improvement is finished. The entire solution of SUALBP-2 is then updated and returned as shown in line 6 and line 7.

The task re-sequencing subproblem on a station is very similar to the travelling salesman problem with precedence constraints, which can be solved with DIDP (Kuroiwa and Beck 2023b). Thus, we formulate each of the re-sequencing problem as a DIDP model. Since the DIDP model is simple, we present it in our online appendix (Zhang and Beck 2024a).

6. Experiments

In this section, we evaluate the performance of our DIDP and CP models compared to the state-of-the-art MIP and FFLBBD models (Esmaeilbeigi et al. 2016, Zohali et al. 2022) using the SBF2 dataset (788 instances) (Scholl et al. 2013).[2]

The dataset includes four setup time levels, defined by the parameter α, which indicates the ratio of average setup time to average task processing time. Larger α values correspond to longer setup times. Zohali et al. (2022) generate four datasets with α values of 0.25, 0.50, 0.75, and 1.00.

The SBF2 dataset, designed for SUALBP-1, provides the optimal number of stations for type-1 but lacks information on the number of stations and optimal cycle time for type-2. Zohali et al. (2022) use $m = \sum_{i \in V} t_i / c$, where c is the pre-defined cycle time from the SBF2 dataset for each instance, making it also applicable for testing SUALBP-2.

Zohali et al. (2022) cluster these instances into four classes:

- Data set A: small (132 instances) with up to 25 tasks.
- Data set B: medium (140 instances) with 28 to 35 tasks.
- Data set C: large (188 instances) with 45 to 70 tasks.
- Data set D: extra-large (328 instances) with 75 to 111 tasks.

For evaluation, we use the fraction of instances solved and proven optimal over time, as well as the fraction over primal integral (Berthold 2013). The primal integral measures the trade-off between solution quality and computational time. For an optimization problem, let s^t be a solution found at time t by an algorithm, s^* be the optimal (or best-known) solution, and c be a function mapping a solution to its cost. The primal gap function p is defined as:

$$p(t) = \begin{cases} 1 & \text{if no } s^t \text{ or } c(s^t)c(s^*) < 0, \\ 0 & \text{if } c(s^t) = c(s^*) = 0, \\ \frac{|c(s^*) - c(s^t)|}{max\{|c(s^*)|, |c(s^t)|\}} & \text{otherwise.} \end{cases} \tag{18}$$

The primal gap ranges from $[0, 1]$, with lower values indicating better performance. We use $p(T)$, the primal gap at the time limit T, to gauge the final solution quality. Let $t_i \in [0, T]$ for

[2] `https://assembly-line-balancing.de/sualbsp/data-set-of-scholl-et-al-2013/`

$i = 1, \dots, L-1$ denote the time points when new incumbent solutions are found, with $t_0 = 0$ and $t_L = T$. The primal integral is defined as:

$$P(T) = \sum_{i=1}^{L} p(t_i - 1) \cdot (t_i - t_{i-1}). \tag{19}$$

The primal integral ranges from $[0, T]$, with lower values indicating better performance. $P(T)$ decreases if a better solution is found within the same computational time or if the same solution cost is achieved more quickly. If an instance is proven infeasible at time t, we set $p(t) = 0$, reflecting the time to prove infeasibility.

For the DIDP models, we use the state-of-the-art solver based on complete anytime beam search (CABS) (Kuroiwa and Beck 2023b, Zhang 1998) in didp-rs v0.4.0.[3] Beam search (Shapiro 1992) is a breadth-first search with the maximum number of nodes at a given depth upper bounded by the *beam width*, b: the top b states according to the heuristic function are expanded at each depth. CABS sequentially performs beam search starting with $b = 1$ and doubling b in each iteration. In a given iteration, beam search may find some feasible solutions as it can reach base case states. CABS proves optimality when it searches all states in the state space and finds no better solution than the incumbent. Some states are pruned via dominance rules and dual bounds without being expanded. Further details of CABS can be found in our online appendix (Zhang and Beck 2024a) and in the literature (Kuroiwa and Beck 2024, Zhang 1998).

We have also tested the local improvement algorithm combined with DIDP models for the original problem and subproblems of SUALBP-2. Since we use CABS as the DIDP solver, we call the entire algorithm *local improvement CABS* (LICABS). For the CP models, we use CP Optimizer 20.1.0 (IBM 2023). For the MIP models, we use Gurobi 9.5.1 (Gurobi Optimization 2021). All the experiments are implemented in Python 3.8. Each instance is run for 1800 seconds on a single thread on a Ubuntu 22.04.2 LTS machine with Intel Core i7 CPU and 16 GB memory. All the code, data, and results are available online (Zhang and Beck 2024b).

6.1. Results for SUALBP-1

The results on SUALBP-1 are shown in Fig. 3 and 4. Better performance is indicated by curves closer to the top left corner of the graphs. The DIDP model outperforms CP and MIP models. More specifically, Fig. 3 shows that the DIDP model finds optimal solutions and proves optimality for more instances in a shorter computation time than the CP and MIP models. In 1 second, DIDP finds

[3] https://didp.ai/

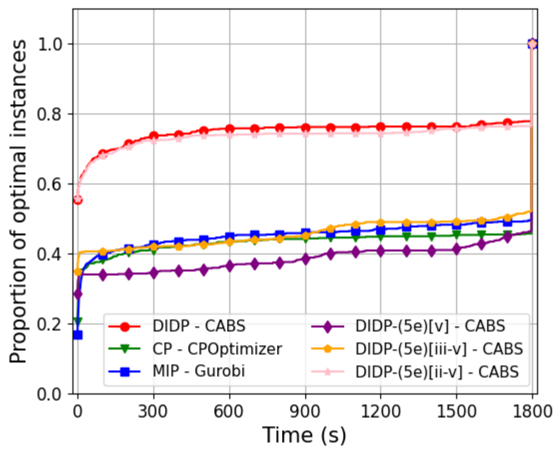

Figure 3 **Ratio of instances solved to optimality over time for SUALBP-1**

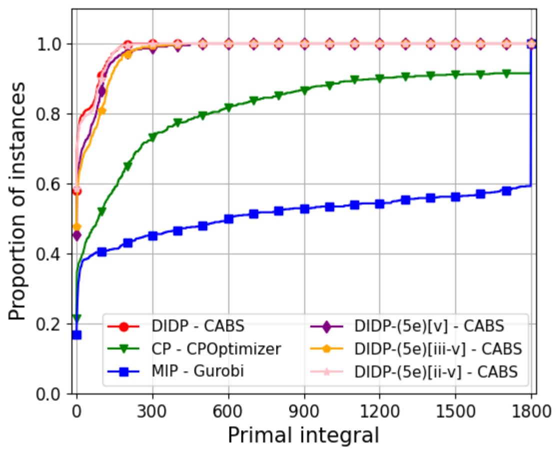

Figure 4 **Ratio of instances over primal integral for SUALBP-1**

and proves optimality on 55% of the instances. MIP and CP cannot achieve the same performance in 1800 seconds. At the 1800 second time limit, DIDP has found and proved optimality for 78% of the problem instances compared to 45% and 50% for CP and MIP, respectively.

In addition to the full DIDP model, we also test three DIDP models with different sets of dual bounds. DIDP-(5e)[v] uses only the 0 dual bound (5e-v), DIDP-(5e)[iii-v] uses the dual bounds from (5e-iii) to (5e-v), and DIDP-(5e)[ii-v] uses the dual bounds from (5e-ii) to (5e-v). While DIDP-(5e)[ii-v] achieves slightly worse performance than the full model, DIDP-(5e)[iii-v] is only competitive with CP and MIP. However, DIDP-(5e)[v] with the 0 dual bound is worse than all compared models. The comparison of the four DIDP models demonstrates the importance of the state-based dual bounds in DIDP models.

Fig. 4 shows that DIDP also finds high-quality solutions faster than CP and MIP models. As is often observed, although the CP model proves fewer optimal solutions than the MIP model, it demonstrates a higher solution quality (Hooker and van Hoeve 2018). However, DIDP outperforms CP and MIP on both measures. The four DIDP variants behave similarly, with the same relative performance ranks as in Fig. 3. This fact indicates that DIDP can find high-quality solutions for SUALBP-1 even without tight dual bounds; in fact, the first feasible solution found by DIDP often has the optimal objective value. CABS then just needs to prove optimality, which is substantially accelerated by the better pruning power brought by tighter dual bounds.

6.2. Results for SUALBP-2

The results on SUALBP-2 are shown in Fig. 5 and 6. The LICABS results are discussed in Section 6.4. As above, better performance is indicated by curves closer to the top and left-side of the

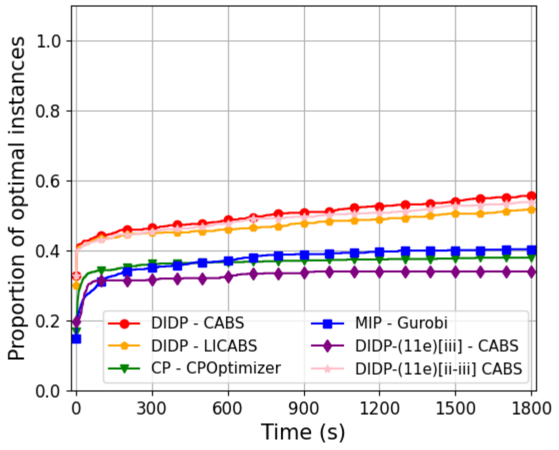

Figure 5 **Ratio of instances solved to optimality over time for SUALBP-2**

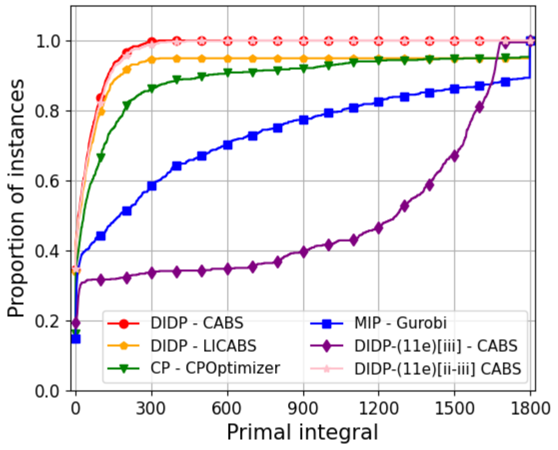

Figure 6 **Ratio of instances over primal integral for SUALBP-2**

graphs. In general, the two graphs demonstrate similar though reduced performance of the three models compared to SUALBP-1. The DIDP model continues to substantially outperform CP and MIP models. Fig. 5 shows that the DIDP model finds optimal solutions and proves optimality for more instances in a shorter computation time than CP and MIP models. In 1 second, DIDP finds and proves optimality on 32% of the instances. CP achieves the same level at 30 seconds and is by that measure 30 times slower. MIP reaches the same level at 120 seconds. At the 1800 second time limit, DIDP has found and proved optimality for 55% of the problem instances compared to 37% and 41% for CP and MIP, respectively.

DIDP-(11e)[iii] is the DIDP model with only the 0 dual bound (11e-iii) and DIDP-(11e)[ii-iii] uses dual bounds (11e-ii) and (11e-iii). While DIDP-(11e)[ii-iii] performs worse than the full DIDP model solved with CABS, it outperforms the full DIDP model solved with LICABS. Similar to the results of SUALBP-1, DIDP-(11e)[iii] with 0 dual bound trails all compared models.

Fig. 6 shows that DIDP finds high-quality solutions faster than CP and MIP models. Although the CP model again proves fewer optimal solutions than the MIP model, it demonstrates a higher solution quality than the MIP model and is almost equal to the DIDP model at the time limit. However, DIDP achieved the same level of quality as CP and MIP in considerably less than 180 and 140 seconds, respectively. DIDP-(11e)[iii] with 0 dual bound is not able to find high-quality solutions for SUALBP-2 in a short time. Different from SUALBP-1, the first few SUALBP-2 solutions found by DIDP do not have optimal objective values. Without tight dual bounds, CABS cannot find high-quality solutions easily due to a lack of promising guidance.

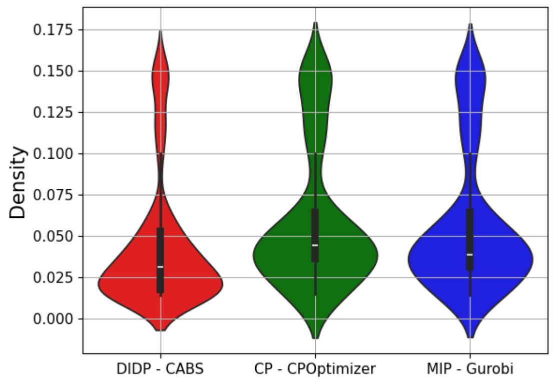

Figure 7 **Density distribution of instances solved to optimality for SUALBP-1**

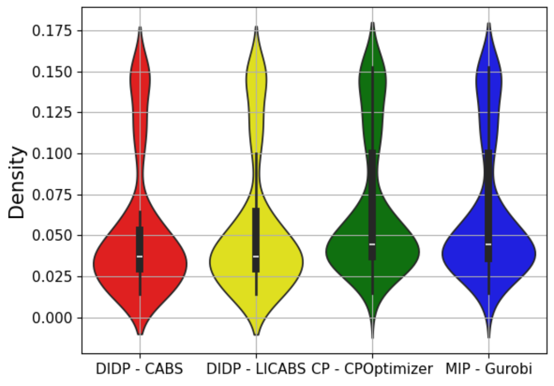

Figure 8 **Density distribution of instances solved to optimality for SUALBP-2**

Overall the experimental results for SUALBP-1 and SUALBP-2 show that DIDP substantially outperforms MIP and CP while CP is competitive with MIP: CP finds better solutions than MIP in a given time but trails MIP in the number of solutions proved optimal over time.

6.3. Algorithm Performance w.r.t Network Density of Instances

We also compare the network density of the instances solve to optimality by DIDP, CP, and MIP, as shown in Fig. 7 and 8. The network density for SUALBP is defined as $\rho = \frac{|\mathcal{E}|}{n(n-1)}$, where $\mathcal{E}$ is the set of all precedence relations and n is the number of tasks. In Fig. 7 and 8, the width of each violin plot at a specific density reflects the proportion of the solved instances with this density, rather the number of the solved instances. Such plots are useful for visualizing the distribution of solved instances for given approach but do not reflect the number of solved instances and thus care should be taken with comparisons across approaches.

All three approaches can handle instances with relatively high density (i.e., more precedence relations), while DIDP can also solve instances without many precedence constraints. This fact indicates the advantage of the DIDP paradigm. For SUALBP-1, as shown in Fig. 7, CP has the highest mean density value, since more precedence constraints enhance the domain filtering by constraint propagation.

6.4. Comparison to Zohali et al. for SUALBP-2

As noted, the state-of-the-art exact algorithm for SUALBP-2 is the *full-featured logic based Benders decomposition* (FFLBBD) approach proposed by Zohali et al. (2022). We were not able to obtain their source code but, as they ran experiments on the same SBF2 data set, we can compare the quality of results to what the authors provided in the online appendix to that paper. The detailed comparison is shown in Table 5. The results of FFLBBD are taken from paper by Zohali et al. (2022) with different hardware, but the same time and memory limits of 1800 seconds and 16 GB. For the overall comparison, DIDP proves 438 instances to optimality while FFLBBD only proves 371. Since these benchmark instances are first investigated by Zohali et al. (2022), according to their results, FFLBBD reports solving 9 instances to optimality that DIDP does not, while DIDP solves 76 instances to optimality that FFLBBD does not. Thus, our DIDP approach solves 67 more instances to optimality than FFLBBD while DIDP closes 76 open instances (i.e., DIDP failed to optimally solve 9 instances which FFLBBD did solve).

For data set A, B, and C, DIDP outperforms FFLBBD in terms of the average optimality gap ('Gap' column), average runtime ('Time' column), the number of instances with feasible solutions found ('Feas' column), and the number of instances with optimal solutions found and proved ('Opt' column). However, for the extra large data set D, FFLBBD is better, though DIDP shows some advantages in proving optimality and solution speed for $\alpha = 0.75$ and $\alpha = 1.00$.

However, our results indicate some problems with the results published in the online appendix of Zohali et al. (2022) where they provide the objective function values for each problem instance. For the small instance `jackson_c=14.alb`, the best lower bound (LB) and upper bound (UB) on the cycle time according to Zohali et al. are both 13. Our DIDP, CP, and MIP models all find a solution with cycle time of 12. We have manually confirmed that the DIDP solution is feasible, implying an error in the LB provided by Zohali et al. For the small instance `jackson_c=7.alb`, the best LB and UB of the cycle time according to Zohali et al. are 8. Our DIDP, CP, and MIP models all find an optimal solution with cycle time being 9. We do not have access to the detailed solution produced by FFLBBD for this instance and so are unable to analyze it further. There are six instances with inconsistent optimal objectives obtained by DIDP/CP/MIP and FFLBBD, as shown in Table 6. In both cases where DIDP finds a better 'optimal' solution, we have verified the solution feasibility. Thus, some of the results reported by Zohali et al. are incorrect.[4]

6.5. Results of the Local Improvement Algorithm for SUALBP-2

Interestingly, as we can see in Fig. 5 and 6, the performance of the local improvement algorithm is worse than our core DIDP model. To explain this observation, we use two metrics: the mean number and mean proportion of incumbent solutions that are improved by local improvement in each problem instance.

[4] Based on recent communication with the authors of Zohali et al., their incorrect results were due to data entry errors. Their corrected entries are consistent with our results on all instances for which their methods and one or more of ours prove optimality.

Table 5 Results of DIDP and FFLBBD: better results are in bold.

Class	α	#	DIDP				FFLBBD			
			Gap	Time	Feas	Opt	Gap	Time	Feas	Opt
A	0.25	33	**0.00**	**0.0007**	**33**	**33**	**0.00**	0.12	**33**	**33**
	0.50	33	**0.00**	**0.0012**	**33**	**33**	**0.00**	0.49	**33**	**33**
	0.75	33	**0.00**	**0.0022**	**33**	**33**	**0.00**	0.64	**33**	**33**
	1.00	33	**0.00**	**0.0038**	**33**	**33**	**0.00**	0.74	**33**	**33**
B	0.25	35	**0.00**	**3**	**35**	**35**	0.02	28	**35**	33
	0.50	35	**0.00**	**4**	**35**	**35**	0.01	42	**35**	34
	0.75	35	**0.00**	**7**	**35**	**35**	0.10	51	**35**	32
	1.00	35	**0.00**	**8**	**35**	**35**	0.11	54	**33**	33
C	0.25	47	**1.47**	**824**	**47**	**34**	2.68	1399	**47**	13
	0.50	47	**3.07**	**922**	**47**	**28**	5.71	1371	**47**	13
	0.75	47	**4.13**	**951**	**47**	**28**	8.17	1350	**47**	14
	1.00	47	**4.99**	**919**	**47**	**27**	10.39	1398	**47**	13
D	0.25	82	11.25	1556	**82**	13	**3.57**	**1440**	**82**	**18**
	0.50	82	14.03	1575	**82**	13	**6.19**	**1551**	**82**	**15**
	0.75	82	17.14	**1616**	**82**	**12**	**9.94**	1620	**82**	11
	1.00	82	18.77	**1627**	**82**	**11**	**13.32**	1639	**82**	9
sum		788			**788**	**438**			**788**	371

Table 6 Instances with different optimal objectives by using DIDP and FFLBBD.

Instance	Class	α	DIDP OptVal	FFLBBD UB	FFLBBD LB
jackson_c=7	A	0.75	9	8	8
jackson_c=14	A	0.50	12	13	13
lutz1_c=2357b	B	0.25	2475	2472	2472
sawyer30_c=54	B	0.75	58	57	57
hahn_c=1806	C	0.25	2830	2848	2848
hahn_c=4676	C	0.25	4847	4798	4798

Table 7 Mean number and proportion of CABS iterations that lead to real improvement.

Instances	All	A	B	C	D
Mean number	0.897	0.167	0.386	0.686	1.530
Mean proportion	0.081	0.037	0.053	0.071	0.117

The results are shown in Table 7. Over all instances, local improvement finds a new incumbent 8.1% of the times that it is called, which appears non-trivial. However, as local improvement solves the subproblems to optimality, it also means that in 91.9% of the calls, our core model had already found the optimal sequence on the station with the maximum cycle time (given the task assignment). As the local improvement is only called when an incumbent is found, on average it only finds 0.897 improving solutions per instance. Thus, local improvement does not bring a performance gain with the runtime spent on it. The mean number and the value of local improvement is increasing as the problem size gets larger, and, in fact, local improvement does find better solutions than our core model for some of the extra large instances.

7. Conclusions

In this paper, we study the assembly line balancing problem with sequence-dependent setup time (SUALBP), where tasks need to be assigned to stations and sequenced within each station, taking into account the processing time of tasks and sequence-dependent setup time between tasks. In type-1 SUALBP, an upper limit of station time is given and the objective is to minimize the number of active stations, while in type-2 SUALBP, the number of stations is fixed and the objective is to minimize the maximum station time.

We developed novel domain-independent dynamic programming (DIDP) and constraint programming (CP) optimization models. The DIDP models are formulated as state-based transition systems and solved with a DIDP solver using complete anytime beam search. We also proposed state-based dual bounds to accelerate the solving process of DIDP models. The CP models adopt a scheduling perspective, using optional interval variables and sequencing constraints.

We compared the performance of the proposed DIDP and CP models with the state-of-the-art MIP models on a diverse set of SUALBP instances from the literature. Experimental results show that the DIDP models are significantly superior to the CP and MIP models in terms of proving optimality and finding high-quality solutions. The CP models prove optimality for slightly fewer instances than MIP but can find higher quality solutions than MIP on average.

For type-2 SUALBP, we compare the proposed DIDP model with the state-of-the-art exact algorithm, the full-featured logic-based Benders decomposition (FFLBBD) (Zohali et al. 2022). DIDP outperforms FFLBBD in terms of proving optimality, especially for small, medium, and large instances. For extra large instances, DIDP is competitive with FFLBBD. In particular, DIDP proves optimality for 67 more instances than FFLBBD and closes 76 open instances, with 9 instances that are proved by FFLBBD not solved optimally by DIDP.

We investigate a local improvement addition to the DIDP model for SUALBP-2, where a station-specific DIDP model seeks to improve new global incumbent solutions by improving the task sequence without changing task-to-station assignments. However, as the re-sequencing rarely improves the solution quality, the overall performance of the algorithm is worse than the core DIDP model.

This work represents an important contribution to exact methods for SUALBP and demonstrates the promising prospect of DIDP for solving complex planning and scheduling problems. Our main directions for future work are the continued exploration of DIDP performance across diverse combinatorial optimization problems as well as a study of solver behavior to develop better DIDP solution approaches.

References

Achterberg T (2009) SCIP: solving constraint integer programs. *Mathematical Programming Computation* 1:1–41.

Akpinar Ş, Baykasoğlu A (2014a) Modeling and solving mixed-model assembly line balancing problem with setups. part ii: A multiple colony hybrid bees algorithm. *Journal of Manufacturing Systems* 33(4):445–461.

Akpinar Ş, Baykasoğlu A (2014b) Modeling and solving mixed-model assembly line balancing problem with setups. part ii: A multiple colony hybrid bees algorithm. *Journal of Manufacturing Systems* 33(4):445–461.

Akpinar S, Elmi A, Bektaş T (2017) Combinatorial benders cuts for assembly line balancing problems with setups. *European Journal of Operational Research* 259(2):527–537.

Andres C, Miralles C, Pastor R (2008) Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. *European Journal of Operational Research* 187(3):1212–1223.

Becker C, Scholl A (2006) A survey on problems and methods in generalized assembly line balancing. *European journal of operational research* 168(3):694–715.

Bergman D, Cire AA, Van Hoeve WJ, Hooker J (2016) *Decision diagrams for optimization*, volume 1 (Springer).

Berthold T (2013) Measuring the impact of primal heuristics. *Operations Research Letters* 41(6):611–614.

Bockmayr A, Kasper T (1998) Branch and infer: A unifying framework for integer and finite domain constraint programming. *INFORMS Journal on Computing* 10(3):287–300.

Boysen N, Schulze P, Scholl A (2022) Assembly line balancing: What happened in the last fifteen years? *European Journal of Operational Research* 301(3):797–814.

Bukchin Y, Raviv T (2018) Constraint programming for solving various assembly line balancing problems. *Omega* 78:57–68.

Çil ZA, Kizilay D, Li Z, Öztop H (2022) Two-sided disassembly line balancing problem with sequence-dependent setup time: A constraint programming model and artificial bee colony algorithm. *Expert Systems with Applications* 203:117529.

Dooms G, Deville Y, Dupont P (2005) Cp (graph): Introducing a graph computation domain in constraint programming. *Principles and Practice of Constraint Programming-CP 2005: 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005. Proceedings 11*, 211–225 (Springer).

Esmaeilbeigi R, Naderi B, Charkhgard P (2016) New formulations for the setup assembly line balancing and scheduling problem. *OR spectrum* 38:493–518.

Gillard X, Schaus P, Coppé V (2021) Ddo, a generic and efficient framework for mdd-based optimization. *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 5243–5245.

Güner F, Görür AK, Satır B, Kandiller L, Drake JH (2023) A constraint programming approach to a real-world workforce scheduling problem for multi-manned assembly lines with sequence-dependent setup times. *International Journal of Production Research* 1–18.

Gurobi Optimization L (2021) Gurobi optimizer reference manual. URL `http://www.gurobi.com`.

Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2):100–107.

Hooker JN (2002) Logic, optimization, and constraint programming. *INFORMS Journal on Computing* 14(4):295–321.

Hooker JN, van Hoeve WJ (2018) Constraint programming and operations research. *Constraints* 23(2):172–195.

IBM (2023) IBM ILOG CPLEX CP Optimizer Https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-cp-optimizer.

Kizilay D (2022) A novel constraint programming and simulated annealing for disassembly line balancing problem with and/or precedence and sequence dependent setup times. *Computers & Operations Research* 146:105915.

Kumar N, Mahto D (2013) Assembly line balancing: a review of developments and trends in approach to industrial application. *Global Journal of Researches in Engineering Industrial Engineering* 13(2):29–50.

Kuroiwa R, Beck JC (2023a) Domain-independent dynamic programming: Generic state space search for combinatorial optimization. *the 33rd International Conference on Automated Planning and Scheduling (ICAPS), 236–244.*

Kuroiwa R, Beck JC (2023b) Solving domain-independent dynamic programming problems with anytime heuristic search. *the 33rd International Conference on Automated Planning and Scheduling (ICAPS), 245–253.*

Kuroiwa R, Beck JC (2024) Domain-independent dynamic programming. *arXiv preprint arXiv:2401.13883* .

Laborie P, Rogerie J, Shaw P, Vilím P (2018) IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG. *Constraints* 23:210–250.

Martino L, Pastor R (2010) Heuristic procedures for solving the general assembly line balancing problem with setups. *International Journal of Production Research* 48(6):1787–1804.

Morrison DR, Sewell EC, Jacobson SH (2014) An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. *European Journal of Operational Research* 236(2):403–409.

Nethercote N, Stuckey PJ, Becket R, Brand S, Duck GJ, Tack G (2007) Minizinc: Towards a standard cp modelling language. *International Conference on Principles and Practice of Constraint Programming*, 529–543 (Springer).

Özcan U (2019) Balancing and scheduling tasks in parallel assembly lines with sequence-dependent setup times. *International Journal of Production Economics* 213:81–96.

Rossi F, Van Beek P, Walsh T (2006) *Handbook of Constraint Programming* (Elsevier).

Şahin M, Kellegöz T (2017) Increasing production rate in u-type assembly lines with sequence-dependent set-up times. *Engineering Optimization* 49(8):1401–1419.

Scholl A, Boysen N, Fliedner M (2013) The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics. *OR spectrum* 35:291–320.

Scholl A, Klein R (1997) Salome: A bidirectional branch-and-bound procedure for assembly line balancing. *INFORMS journal on Computing* 9(4):319–334.

Seyed-Alagheband S, Ghomi SF, Zandieh M (2011) A simulated annealing algorithm for balancing the assembly line type ii problem with sequence-dependent setup times between tasks. *International Journal of Production Research* 49(3):805–825.

Shapiro SC (1992) Encyclopedia of artificial intelligence second edition. *New Jersey: A Wiley Interscience Publication* .

Tang Q, Liang Y, Zhang L, Floudas CA, Cao X (2016) Balancing mixed-model assembly lines with sequence-dependent tasks via hybrid genetic algorithm. *Journal of Global Optimization* 65(1):83–107.

Wee T, Magazine MJ (1982) Assembly line balancing as generalized bin packing. *Operations Research Letters* 1(2):56–58.

Yang W, Cheng W (2020) Modelling and solving mixed-model two-sided assembly line balancing problem with sequence-dependent setup time. *International Journal of Production Research* 58(21):6638–6659.

Yolmeh A, Kianfar F (2012) An efficient hybrid genetic algorithm to solve assembly line balancing problem with sequence-dependent setup times. *Computers & industrial engineering* 62(4):936–945.

Zhang J, Beck JC (2024a) Appendix to Domain-Independent Dynamic Programming and Constraint Programming Approaches for Assembly Line Balancing Problems with Setups `https://github.com/INFORMSJoC/2024.0603/blob/main/IJOC_SUALBP_Appendix.pdf`.

Zhang J, Beck JC (2024b) Domain-Independent Dynamic Programming and Constraint Programming Approaches for Assembly Line Balancing Problems with Setups URL `http://dx.doi.org/10.1287/ijoc.2024.0603.cd`, available for download at https://github.com/INFORMSJoC/2024.0603.

Zhang W (1998) Complete anytime beam search. *AAAI/IAAI*, 425–430.

Zohali H, Naderi B, Roshanaei V (2022) Solving the type-2 assembly line balancing with setups using logic-based benders decomposition. *INFORMS Journal on Computing* 34(1):315–332.