Transition Dominance in Domain-Independent Dynamic Programming

J. Christopher Beck
Department of Mechanical and Industrial Engineering, University of Toronto

Ryo Kuroiwa
National Institute of Informatics, Tokyo, Japan
The Graduate School of Advanced Studies, SOKENDAI, Tokyo, Japan

Jimmy H.M. Lee
Department of Computer Science and Engineering, The Chinese University of Hong Kong

Peter J. Stuckey
Monash University, Department of Data Science and AI, Australia
ARC Training Centre OPTIMA, Australia

Allen Z. Zhong
Monash University, Department of Data Science and AI, Australia
ARC Training Centre OPTIMA, Australia

Abstract

Domain-independent dynamic programming (DIDP) is a model-based paradigm for dynamic programming (DP) that enables users to define DP models based on a state transition system. Heuristic search-based solvers have demonstrated strong performance in solving combinatorial optimization problems. In this paper, we formally define *transition dominance* in DIDP, where one transition consistently leads to better solutions than another, allowing the search process to safely ignore dominated transitions. To facilitate the efficient use of transition dominance, we introduce an interface for defining transition dominance and propose the use of *state functions* to cache values, thereby avoiding redundant computations when verifying transition dominance. Experimental results on DP models across multiple problem classes indicate that incorporating transition dominance and state functions yields a 5 to 10 times speed-up on average for different search algorithms within the DIDP framework compared to the baseline.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases Dominance, Dynamic Programming, Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.CP.2025.6

Supplementary Material *Software (Source Code)*: `https://github.com/domain-independent-dp/didp-rs/releases/tag/transition-dominance-cp25`

Funding This work was supported by the Australian Research Council OPTIMA ITTC IC200100009, a General Research Fund (CUHK14206321) by the Hong Kong University Grants Committee, a Direct Grant by CUHK, and the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

Dynamic programming [2] is a classical approach to solving combinatorial optimization problems. While it is often the most efficient way of solving many problems, traditionally dynamic programming solutions are hard coded for each problem of interest. *Domain-Independent Dynamic Programming* (DIDP) [13] is a new paradigm to allow the statement of the Bellman equations defining a combinatorial opmisation problem, independent of the techniques used to solve these equations, separating the specification of the model from the solving, as in other complete solving approaches such as constraint programming (CP) and mixed integer programming (MIP). This separation enables easy experimentation with different search algorithms for the same problem domain and enables research on different DP models for a given problem. Empirical evaluation demonstrates the framework outperforms CP and MIP on multiple problem classes, closing a number of open problem instances [14].

The *Dynamic Programming Description Language* (DyPDL) [13] is a powerful modelling language for DIDP that facilitates the expression of dynamic programming (DP) models. Beyond the problem components, DyPDL also supports the modelling of redundant information, such as state dominance and dual bounds, to improve the performance of various solving approaches. In this paper, we extend this framework by formally defining *transition dominance*, redundant information that indicates that certain transitions can be safely ignored if a state satisfies a certain condition. By incorporating transition dominance into DIDP, we make these advanced techniques accessible in a solver-independent manner. Deriving dominances in different problem domains is a challenging task that requires a deep understanding of problem structures. Our case studies demonstrate that the insights of identifying new dominances can be applied to similar problem domains, enabling the discovery of transition dominance from common substructures. Our key contributions are as follows:

1. We formally define the concept of *transition dominance* for dynamic programming models.
2. We introduce a new component into DyPDL to allow users to model transition dominance effectively and *state functions* to avoid redundant computations.
3. We present new transition dominances for several combinatorial optimization problems.
4. We demonstrate that using transition dominance and state functions substantially improve the performance of various search algorithms with 5 to 10 times speed-ups.

2 Background

A DyPDL model is a tuple $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, where $\mathcal{V}$ is a set of *state variables*, S^0 is the *target state*, $\mathcal{T}$ is a set of *transitions*, $\mathcal{B}$ is a set of *base cases*, and $\mathcal{C}$ is a set of *state constraints*. A state variable can be an *element*, *set*, or *numeric variable*. A state $S \in \mathcal{D}$ is a tuple that assigns values to state variables in $\mathcal{V}$, where the value of a variable $v \in \mathcal{V}$ is denoted by $S[v]$.

Expressions are used in transitions, base cases, and state constraints to describe the computation of a value using state variables and constants. When an expression e is evaluated given a state S, it returns a value $e(S)$. A *numeric expression* returns a numeric value $e(S) \in \mathbb{Q}$. It can refer to a numeric constant or variable, use arithmetic operations, and take the cardinality of a set expression. An *element expression* returns a nonnegative integer $e(S) \in \mathbb{Z}_0^+$, while a *set expression* returns a set $e(S) \in 2^{\mathbb{Z}_0^+}$. A condition *cond* is a function mapping a state S to a Boolean value $c \in \{\top, \bot\}$. We say $S \models \mathit{cond}$ if $c = \top$, and $S \models C$ for a set of conditions C if $S \models \mathit{cond}$ for all $\mathit{cond} \in C$. A condition can refer to a Boolean constant, compare two elements or numeric expressions, or check whether an element is included in a set. The conjunction and disjunction of two conditions are also conditions. Base cases in $\mathcal{B}$ and state constraints in $\mathcal{C}$ are conditions. When a base case $B \in \mathcal{B}$ is satisfied by a state, the state is called a base state, and the set of all base states is denoted as $\mathcal{S}_{\mathcal{B}}$. We assume that a function $\mathsf{base\_cost} : \mathcal{S}_{\mathcal{B}} \mapsto \mathbb{Q}$ is defined, which returns a numeric value $\mathsf{base\_cost}(S)$ given a base state S.

A transition $\tau \in \mathcal{T}$ has *effect* $\mathrm{eff}_\tau : \mathcal{D} \mapsto \mathcal{D}$ which maps a state S to another state $S[\![\tau]\!]$, and *cost* $\mathsf{cost}_\tau : \mathbb{R} \cup \{\infty\} \times \mathcal{D} \mapsto \mathbb{R} \cup \{\infty\}$ which maps a real value r and a state S to a value $\mathrm{cost}_\tau(r, S)$. *Preconditions* pre_τ are conditions on state variables, and τ is *applicable* in a state S only if all preconditions are satisfied, denoted by $S \models \mathrm{pre}_\tau$.

Solving a DyPDL model requires finding a sequence of transitions with the optimal cost to transform the target state S^0 into a base state. Let $\sigma = \langle \sigma_1, \ldots, \sigma_m \rangle$ be a sequence of transitions applicable from S, i.e., $S \models \text{pre}_{\sigma_1}$ and $S^i \models \text{pre}_{\sigma_{i+1}}$ where $S^1 = S[\![\sigma_1]\!]$ and $S^{i+1} = S^i[\![\sigma_{i+1}]\!]$. Then, σ is an S-solution if S^m is a base state, S and each S^i with $i = 1, ..., m$ satisfy state constraints, and S and S^i with $i < m$ are not base states. If S is a base state, we assume that an empty sequence $\langle\rangle$ is an S-solution. The cost of an S-solution σ is defined recursively as

- if $\sigma = \langle\rangle$, $\mathsf{solution\_cost}(\sigma, S) = \mathsf{base\_cost}(S)$
- otherwise, $\mathsf{solution\_cost}(\sigma, S) = \mathsf{cost}_{\sigma_1}(\mathsf{solution\_cost}(\langle \sigma_2, ..., \sigma_m \rangle, S[\![\sigma_1]\!]), S)$

An *optimal solution* for minimization is an S-solution whose cost is less than or equal to the cost of any S-solution. Let V be a function of a state S that returns ∞ if there are no S-solution or the cost of an optimal S-solution otherwise.

In this paper, we assume that a DyPDL model is both *finite* and *acylic*, and the cost expression satisfies the Principle of Optimality [2]. Under these assumptions, $V(S)$ can be computed by Bellman equation [15]:

$$V(S) = \mathsf{base\_cost}(S), \qquad \text{if } S \in \mathcal{S}_\mathcal{B} \tag{1a}$$

$$V(S) = \infty, \qquad \text{else if } S \not\models \mathcal{C} \tag{1b}$$

$$V(S) = \min_{\tau \in \mathcal{T}(S)} \mathsf{cost}_\tau(V(S[\![\tau]\!]), S) \qquad \text{else} \tag{1c}$$

State dominance is a type of redundant information useful for improving the solving efficiency of DyPDL models.

▶ **Definition 1.** *A state S* dominates *another state S', i.e. $S \preceq S'$, if and only if there exists an S-solution σ for any S'-solution σ' such that $cost_\sigma(S) \leq cost_{\sigma'}(S')$ and $|\sigma| \leq |\sigma'|$.*

In practice, it is challenging to detect and exploit state dominance, and therefore the DyPDL formulation focuses on its approximation defined by *resource state variables* with additional *preference*. A state S is *preferred over* (or approximately dominates) another state S', denoted as $S \preceq_a S'$, iff $S[v] \geq S'[v]$ for resource variables where greater is preferred, $S[v] \leq S'[v]$ for resource variables where less is preferred, and $S[v] = S[v']$ for non-resource variables.

3 Transition Dominance in DIDP

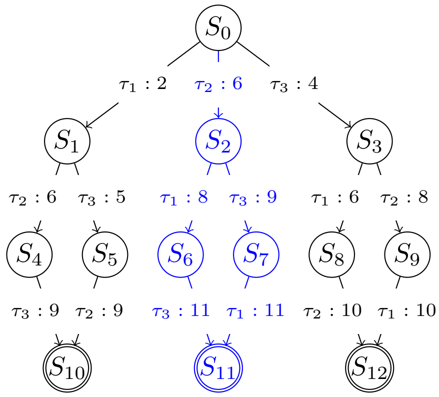
In this paper, we enhance the framework of DIDP by introducing *transition dominance*. State dominance falls into the category of *memory-based* dominance rules [22], where an unexpanded state is compared with previously generated states and pruned if it is shown to be dominated. In contrast, transition dominance is a type of *non-memory-based dominance* that implies the existence of a better solution without requiring additional memory.

▶ **Definition 2.** *Transition dominance at a state S is a relation defined over $\mathcal{T}(S)$ such that τ dominates τ' in S, denoted as $\tau \preceq^S \tau'$, implies that $cost_\tau(V(S[\![\tau]\!]), S) \leq cost_{\tau'}(V(S[\![\tau']\!]), S)$.*

It is straightforward to show that transition dominance is a transitive relation. When a transition dominance relation in a state S is *irreflexive*, i.e., $\tau \not\preceq^S \tau$, we denote the dominance relation by $\prec^S$. If transition dominance relations in all states are irreflexive, removing dominated transitions for all states preserves the optimal cost for a DyPDL model.

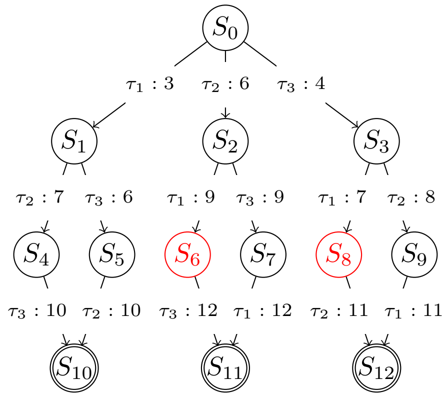
▶ **Proposition 3.** *Let $\mathcal{T}^*(S) = \{\tau' \in \mathcal{T}(S) \mid \nexists \tau \in \mathcal{T}(S), \tau \prec^S \tau'\}$ be the set of non-dominated transitions for a state S where $\prec^S$ is an irreflexive transition dominance relation in a state S. The optimal cost of the DyPDL model $\langle \mathcal{V}, \mathcal{S}^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ is equivalent to that of $\langle \mathcal{V}, \mathcal{S}^0, \mathcal{T}^*, \mathcal{B}, \mathcal{C} \rangle$.*

Job	Release(r_i)	Processing(p_i)
1	0	2
2	2	4
3	1	3



(a) Example with transition dominance

Job	Release(r_i)	Processing(p_i)
1	0	**3**
2	2	4
3	1	3



(b) Example with state dominance

Figure 1 Example job sequencing instances with transition dominance and state dominance.

The proof, inspired by Chu and Stuckey [4], is in Appendix A.

Note that τ dominates τ' in state S, is different from state dominance $S[\![\tau]\!] \preceq S[\![\tau']\!]$. Transition dominance $\tau \preceq^S \tau'$ holds as long as $\mathsf{cost}_\tau(V(S[\![\tau]\!]), S) \leq \mathsf{cost}_{\tau'}(V(S[\![\tau']\!]), S)$, whereas state dominance between $S[\![\tau]\!]$ and $S[\![\tau']\!]$ requires $V(S[\![\tau]\!]) \leq V(S[\![\tau']\!])$. These two inequalities do not necessarily imply each other, as their relationship depends on the specific cost expressions. In practice, state dominance is approximately determined by comparing the values of state variables between two states, whereas transition dominance depends only on the values of variables within a single state.

► **Example 4.** Consider a sequencing problem with three jobs, where each job has a release time and a processing time, and can only be processed after its release time. The objective is to find a sequence of jobs that minimizes the total completion time. We can model this problem using a set variable R to represent the remaining jobs and an integer resource variable t to represent the current time. A transition τ_i for each job i schedules the job next, with the precondition $i \in R$. The optimal value can be computed as follows:

$$\begin{aligned} &V(R,t) = 0, && \text{if } R = \emptyset \\ &V(R,t) = \min_{i \in R}\{\max(t, r_i) + p_i + V(R \setminus \{i\}, \max(t, r_i) + p_i)\} && \text{else} \end{aligned}$$

where $\max(t, r_i) + p_i$ is the completion time for job i given that the current time is t. Figure 1 gives two example instances and their complete state transition graphs. Each edge in the transition graph is labeled with a transition and its corresponding value of $\max(t, r_i) + p_i$. Base states are represented with a double-circle.

Figure 1a illustrates the effect of *transition dominance*, where pruned solutions are highlighted in blue. Transition τ_1 dominates τ_2 in state S_0 because the release time of job 2 is not earlier than the time when job 1 can be fully processed. Thus, directly selecting τ_1 from S_0 is always preferable, leading to the pruning of all S_0-solutions that start with τ_2. Note that transition dominance concerns solutions originating from a single state but differing in their initial transitions.

Figure 1b illustrates the effect of state dominance, where the processing time of job 1 is modified to 3, and so τ_1 no longer dominates τ_2. Note that t is a resource variable where a smaller value is preferred, which defines the state dominance criterion: if a state S has the same set of remaining jobs as another state S', but a smaller or equal current time, i.e., $S[R] = S'[R]$ and $S[t] \leq S'[t]$, then S dominates S'. A close examination reveals that S_4 dominates S_6, S_5 dominates S_8, and S_9 dominates S_7. Assuming that the state graph is explored using depth-first search, proceeding from left to right, solutions passing through S_6 and S_8, highlighted in red, are pruned by state dominance in this example. As shown, pruning by state dominance and by transition dominance operate differently. ◀

4 Modelling Transition Dominance in DIDP

Theoretically, transition dominance is defined for each state, but transition dominance rules that are valid for a set of states satisfying a certain condition are more interesting in practice. In this section, we introduce two constructs to facilitate modelling transition dominance.

4.1 An Interface for Transition Dominance

Exploiting transition dominance avoids the generation of successor states by dominated transitions. A direct way to specify transition dominance is to add additional preconditions for transitions. Modelers first identify a condition $c_{\tau \preceq \tau'}$ on states such that if a state S satisfies it, then τ dominates τ' in S, i.e., $S \models c_{\tau \preceq \tau'} \to \tau \preceq^S \tau'$. Then, the negation $\neg(c_{\tau \preceq \tau'} \land pre_\tau)$ is added to the preconditions for the dominated transition τ', which states that τ' is applicable only if either transition τ is not applicable or the condition for transition dominance $\tau \preceq^S \tau'$ is not satisfied. Otherwise, τ' can be pruned since τ is a better applicable transition. In DP-based tools with non-declarative APIs like ddo [10] and CODD [20], transition dominance can be implemented by manually checking these preconditions during successor generation.

▶ **Example 5.** Consider the transition dominance in Example 4 again. If there are two jobs i and j such that the time after scheduling i and j consecutively is no greater than that after scheduling j only in any state, then scheduling i first is no worse than scheduling j first. Formally, for any state S, if $precede(i, j) = \max(S[t], r_i) + p_i \leq r_j$ is true, then taking τ_i at state S is no worse than taking τ_j. To fully utilize the power of transition dominance, each transition τ_j should be compared against all other transitions to determine whether there is a transition τ_i which dominates τ_j. Therefore, we need to augment the precondition of τ_j with $\land_{i \neq j} \neg\{precede(i, j) \land (i \in R)\}$. ◀

Directly modelling transition dominance through preconditions introduces two key challenges. First, it conflates the redundant information of transition dominance with the actual preconditions of transitions, making it difficult for solvers to treat these distinct model elements differently. Second, multiple conditions for dominance require careful handling of tie-breaking for symmetric transitions that dominate each other in a state.

▶ **Example 6.** Consider again the job sequencing problem in Example 4. If the current time t is greater than the release times for two jobs i and j and $p_i \geq p_j$, then transition τ_i dominates another transition τ_j. This is a special case of Proposition 16. However, while the dominance is valid if the jobs have *equal* processing time, adding preconditions for both jobs may make a satisfiable instance unsatisfiable as both transitions become inapplicable.

Multiple conditions for dominance may form reflexive transition dominance relations in some states, violating the requirement in Proposition 3 to preserve the optimality. To resolve this issue, modelers would need to carefully implement tie-breaking between symmetric transitions, adding unnecessary complexity to the model.

To address these challenges, we introduce a new function in DIDPPy, a Python interface for using DIDP, that allows users to explicitly define transition dominance. The function `model.add_transition` returns a unique identifier for the newly added transition that can later be used to retrieve the transition and define transition dominance relationships. The function `model.add_transition_dominance` is provided for this purpose and takes three arguments: the identifiers of the dominating and dominated transitions, and the condition for transition dominance. The following shows a model for the problem in Example 5.

Listing 1 Sample model for job sequencing with transition dominance

```
import didppy as dp, itertools
r = [0, 2, 1]
p = [2, 4, 3]
all_jobs = [0, 1, 2]
ids=[]

model = dp.Model()
jobs = model.add_object_type(number=3)
R = model.add_set_var(target=all_jobs, object_type=jobs)
t = model.add_int_resource_var(target=0, less_is_better=True)
model.add_base_case([R.is_empty()])
for i in all_jobs:
    tran = dp.Transition(
        name="schedule job{}".format(i),
        cost=(dp.max(t, r[i]) + p[i]) + dp.IntExpr.state_cost(),
        effects=[(R, R.remove(i)), (t, dp.max(t, r[i]) + p[i])],
        preconditions=[R.contains(i)],
    )
    id = model.add_transition(tran)
    ids.append(id)

for i, j in itertools.permutations(all_jobs, 2):
    if i != j:
        model.add_transition_dominance(
            ids[i], ids[j], [dp.max(t, r[i]) + p[i] <= r[j]]
        )
```

We formalize the entity defined with our interface as a *transition dominance rule.*

▶ **Definition 7.** *A transition dominance rule is a triple* (τ, τ', c) *such that* $S \models c \rightarrow \mathsf{cost}_\tau(V(S[\![\tau]\!]), S) \leq \mathsf{cost}_{\tau'}(V(S[\![\tau']\!]), S)$.

Suppose D *is a set of transition dominance rules. The transition dominance graph* $G(S) = (N, E)$ *for a state* S *is a directed graph where the set of nodes is* $N = \mathcal{T}(S)$ *and a directed edge* $(\tau, \tau') \in E$ *exists if* $\exists(\tau, \tau', c) \in D$ *with* $S \models c$.

A transition dominance rule provides a partial description of transition dominance relations across all states. Using the transition dominance graph $G(S)$, we consider the transitive closure of the binary relation induced by a given set of transition dominance rules. Concretely, the binary relation $\preceq^S$ over $\mathcal{T}(S)$ *derived from* $G(S)$ is defined such that there is a path from τ to τ'.

▶ **Proposition 8.** *The relation $\preceq^S$ derived from a transition dominance graph is a transition dominance relation for S.*

Proof. For a pair of transitions such that $\tau \preceq^S \tau'$, there exists a path $(\tau_1, ..., \tau_m)$ in the transition dominance graph where $\tau = \tau_1$ and $\tau' = \tau_m$. By Definition 7, $\mathsf{cost}_{\tau_i}(V(S[\![\tau_i]\!]), S) \leq \mathsf{cost}_{\tau_{i+1}}(V(S[\![\tau_{i+1}]\!]), S)$ for $i = 1, .., m-1$. Thus, $\mathsf{cost}_{\tau}(V(S[\![\tau]\!]), S) \leq \mathsf{cost}_{\tau_2}(V(S[\![\tau_2]\!]), S) \leq ... \leq \mathsf{cost}_{\tau'}(V(S[\![\tau']\!]), S)$, which satisfy the requirement of a transition dominance relation. ◀

While irreflexivity is required to ensure there exists at least one optimal solution as shown in Proposition 3, a user may define symmetric transition dominance rules, e.g., (τ, τ', c) and (τ', τ, c'), such that there exists a state S where $S \models c \wedge c'$. To enforce irreflexivity in the transition dominance relation derived from the transition dominance graph while pruning as many dominated transitions as possible, we identify all strongly connected components (SCCs) and construct a *contracted transition dominance graph* as follows.

▶ **Definition 9.** *Given the transition dominance graph $G(S)$ for a state S, a directed graph $G'(S)$ is a contracted transition dominance graph if the set of nodes is $\mathcal{T}(S)$, and edges are constructed with the following procedure:*

1. *In each SCC of $G(S)$, select a representative node r, and add edge (r, v) to $G'(S)$ for each node $v \neq r$ in the SCC.*
2. *For each edge (u, v) in $G(S)$ such that u and v are in different SCCs, add an edge (r, r') to $G'(S)$ where r and r' are the representative nodes of the SCCs to which u and v belong.*

▶ **Proposition 10.** *The binary relation $\preceq^S$ derived from a contracted transition dominance graph is an irreflexive transition dominance relation.*

Proof. Let $G(S)$ be the transition dominance graph for a state S, and let $G'(S)$ be the contracted transition dominance graph. We show that if $G'(S)$ has a path from τ to τ', then $G(S)$ also has a path from τ to τ'. Then, by Proposition 8, the binary relation derived from $G'(S)$ is a transition dominance relation. Consider an arbitrary edge (u, v) on a path from τ to τ' in $G'(S)$. We show that $G(S)$ has a path from u to v, and thus, by concatenating such paths, we can find a path from τ to τ' in $G(S)$. If u and v belong to the same SCC in $G(S)$, then there is a path from u to v in $G(S)$. If u and v belong to different SCCs in $G(S)$, then by the construction of $G'(S)$, they represent their respective SCCs. Since $G'(S)$ contains the edge (u, v), there exists an edge (u', v') in $G(S)$ where u' belongs to the SCC of u and v' belongs to the SCC of v. Moreover, $G(S)$ has a path from u to u' and a path from v' to v. Thus, a path from u to v exists in G that includes (u', v').

Next, we show that $G'(S)$ is acyclic, which implies that the derived binary relation over $\mathcal{T}(S)$ is irreflexive. Assume for contradiction that $G'(S)$ contains a cycle, i.e., there exists a path from u to v and a path from v to u in $G'(S)$. By the previous argument, $G(S)$ must then contain a path from u to v and a path from v to u, meaning that u and v belong to the same SCC in $G(S)$. However, since u and v have outgoing edges in $G'(S)$, they must be representative nodes of different SCCs, contradicting the fact that they are in the same SCC. Thus, $G'(S)$ cannot contain a cycle. ◀

Finally, we show the maximality of a contracted transition dominance graph, i.e., we cannot use more transition dominance rules while keeping irreflexivity.

▶ **Proposition 11.** *Let $G(S)$ be the transition dominance graph for a state S and $G'(S)$ be a contracted transition dominance graph. Then, there does not exist an acyclic graph $G''(S)$ with the set of nodes $\mathcal{T}(S)$ satisfying all of the following properties:*

1. *If $G''(S)$ has a path from τ to τ', then $G(S)$ also has a path from τ to τ'.*
2. *If a node τ has an incoming edge in $G'(S)$, then τ does so also in $G''(S)$.*
3. *There exists a node τ that has an incoming edge in $G''(S)$ but not in $G'(S)$.*

Proof. Assume that an acyclic graph $G''(S)$ with the described properties exists. Based on the third property, let τ be a node that has an incoming edge in $G''(S)$ but not in $G'(S)$. By construction of $G'(S)$, τ must be the representative node of an SCC in $G(S)$. Let N' be the set of nodes in this SCC. In $G(S)$, each node $v \in N'$ does not have incoming edges from nodes in different SCCs; otherwise, an edge from a representative node in a different SCC to τ should have been added to G'.

In $G'(S)$, since τ is the representative node of N', each node $v \in N' \setminus \{\tau\}$ has incoming edge (τ, v). By the second property, in $G''(S)$, each node $v \in N' \setminus \{\tau\}$ has an incoming edge. Since τ also has an incoming edge in $G''(S)$, all nodes in N' have incoming edges in $G''(S)$. By the previous paragraph, an incoming edge to each $v \in N'$ must be from a node in N'. In $G''(S)$, since each node in N' has an incoming edge from another node in N', there must be a cycle, contradicting our assumption. ◀

Note that if two transitions dominate each other, the choice of which dominance to enforce can impact search efficiency. However, by definition, our model provides no basis for preferring one over the other. In practice, we use Tarjan's algorithm to detect all SCCs and keep the first node visited by depth-first search as the representative node of each SCC.

Note that using the transition dominance interface and preconditions offer different advantages in terms of computational efficiency. Suppose that a transition τ is dominated by τ' if $S \models c_{\tau' \preceq \tau}$ and by τ'' if $S \models c_{\tau'' \preceq \tau}$. Given a state S with $\mathcal{T}(S) = \{\tau, \tau', \tau''\}$, the transition dominance interface requires evaluating $c_{\tau' \preceq \tau}$ and $c_{\tau'' \preceq \tau}$ and performing SCC extraction to construct a contracted transition dominance graph. However, when transition dominance is modeled with preconditions, we can avoid evaluating $c_{\tau'' \preceq \tau}$ once we detect $S \models c_{\tau' \preceq \tau}$. In contrast, using preconditions requires restricting conditions for transition dominance for tie-breaking, which may result in lost opportunities to exploit certain transition dominances within a state. Therefore, selecting the appropriate method depends on the trade-off between computational overhead and the effectiveness of transition dominance exploitation.

4.2 Avoiding Redundant Computation using State Functions

Modelling transition dominance typically requires pairwise comparisons of all applicable transitions of a state. To prevent the generation of dominated solutions, the condition $c_{\tau \preceq \tau'}$ should be evaluated for all pairs (τ, τ') of transitions where τ potentially dominates τ'. However, evaluating these conditions often results in redundant computations.

▶ **Example 12.** Consider the problem in Example 5 and its corresponding model in Listing 1. The expression $\max(t, r_i) + p_i$ for all transitions, which is represented by the `next_time` array in the model, appears in the cost expression, the effects of the transition τ_i for job i, and the condition for transition dominance when comparing τ_i with other transitions. Suppose there are n jobs in total. The expression $\max(t, r_i) + p_i$ needs to be evaluated $O(n)$ times when generating applicable transitions and removing dominated transitions. By caching the values of such redundant evaluations, the number of times to evaluate such expression can be reduced to $O(1)$ for each transition. ◀

To address this observation, we introduce *state functions* in DyPDL. State functions are defined by expressions using state variables and can be used in preconditions, effects, and cost expressions, just as state variables. During the solving process, state functions are lazily computed and cached. The following shows the usage in the job sequencing example.

```
next_time = [dp.max(t, r[i]) + p[i] for i in all_jobs]
next_time_sf = [dp.add_int_state_fun(next_time[i]) for i in all_jobs]
```

To use state functions in Listing 1, we define state functions `next_time_sf` for all transitions τ_i and replace all occurrences of `dp.max(t, r[i]) + p[i]` with `next_time_sf[i]`. When checking if the condition to test whether τ_0 dominates τ_1, the expression `next_time[0]` is computed and cached, and for all other comparison between τ_0 and τ_j for $j = 2, \ldots, n$, the value is reused and therefore redundant computation of `next_time[0]` is avoided. State functions are similar to axioms in PDDL [28], which are derived predicates whose truth is inferred from values of some basic predicates, while results of state functions can be Boolean, integer, or set values.

As we will show in the experimental evaluation, using state functions is sometimes crucial in exploiting transition dominance efficiently. Additionally, state functions also avoid duplicate recomputation in the original models for problems such as Talent Scheduling [25] and OPTW [11]. More detailed descriptions of problems and models are in Appendix B.

5 Case Studies

In this section, we demonstrate the applicability of transition dominance across various optimization problems. For each problem, we describe its DyPDL model and the identified transition dominance. All proofs of propositions in this section are in Appendix A.

5.1 Aircraft Landing

The aircraft landing problem [18] involves scheduling the landing of a set of aircraft on multiple runways to minimize the delay from the target landing times. The aircraft are partitioned into multiple classes, and it is assumed that each class follows a first-come-first-serve sequence according to the target landing time. The decision at a state, then, is to determine which class of aircraft should land next on each runway. The model uses state $S = (\vec{n}, \vec{l}, \vec{c})$, where $n_i \in \vec{n}$ is the index of the next aircraft for each class i, $l_j \in \vec{l}$ is the most recent landing time for runway j, and $c_j \in \vec{c}$ is the class of the most recently landed aircraft for runway j. The target and latest landing times of the next aircraft of class i are t_{i,n_i} and p_{i,n_i}, respectively.

Landing two aircrafts of class i and i' consecutively on the same runway must respect the minimum separation time $sep_{i,i'}$. A transition $\tau_{i,j}$ lands the next aircraft of class i on runway j with the delay $d(S,i,j) = (l_j + \text{sep}_{c_j,i} - t_{i,n_i})^+$, where $(x)^+$ is 0 if $x < 0$ and x otherwise. This transition is applicable only when $n_i > 0$ and the actual landing time is no later than p_{i,n_i}. The optimal value is computed as:

$$\begin{aligned} &V(S) = 0, && \text{if } n_i = 0, \forall i \\ &V(S) = \min_{\tau_{i,j} \in \mathcal{T}(S)} \{d(S,i,j) + V(S[\![\tau_{i,j}]\!])\} && \text{else} \end{aligned}$$

where transition $\tau_{i,j}$ updates n_i to $n_i - 1$, l_j to $\max(l_j + \text{sep}_{c_j,i}, t_{i,n_i})$, and c_j to i.

We follow Coppé et al. [6] to add state dominance that $\vec{l}$ are integer resource variables where less is preferred. We define transition dominance where landing the aircraft for class i is better than landing another class i' on the same runway if the former can be landed before

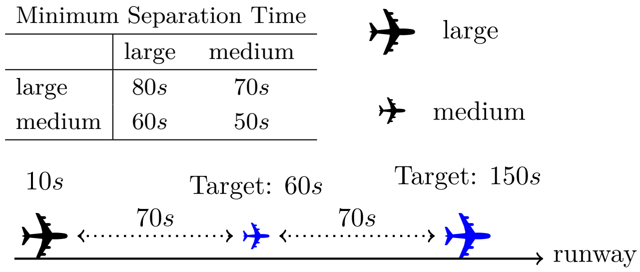


Figure 2 An example instance of aircraft landing.

$t_{i',n_{i'}} - \text{sep}_{i,i'}$. Figure 2 shows an example with two aircraft classes. A large aircraft has landed on the runway at $10s$, and the target landing times for the next medium and large aircraft are $60s$ and $150s$ respectively. The target landing time of the next large aircraft is so late that the next medium aircraft can land first without delaying the large aircraft.

▶ **Proposition 13.** *Suppose minimum separation times satisfy the triangle inequality:* $\text{sep}_{i,j} + \text{sep}_{j,k} \geq \text{sep}_{i,k}$ *for any classes* i, j, k*. If* $\tau_{i,j}, \tau_{i',j} \in \mathcal{T}(S)$ *satisfy*

$$\max(l_j + \text{sep}_{c_j,i}, t_{i,n_i}) \leq t_{i',n_{i'}} - \text{sep}_{i,i'}$$

at the current state S*, then* $\text{cost}_{\tau_{i,j}}(V(S[\![\tau_{i,j}]\!]), S) \leq \text{cost}_{\tau_{i',j}}(V(S[\![\tau_{i',j}]\!]), S)$. ◀

The transition dominance described above schedules a task, such as landing an aircraft, before the start of another task. This type of transition dominance also applies to other problems, such as the problem in Example 4, the Orienteering Problem with Time Windows (OPTW) and the Travelling Salesman Problem with Time Windows and Makespan Objective (TSPTW-M). Detailed problem descriptions are provided in Appendix B.

5.2 Graph-Clear

The Graph-Clear problem arises from multi-robot surveillance tasks [12]. The problem aims to find a sequence of steps to decontaminate ("sweep") nodes in an undirected graph (N, E) with all nodes being initially contaminated. The objective is to minimize the maximum number of robots used over all steps in a given indoor environment modelled as a graph. Each node $i \in N$ can be swept using a_i robots, and each edge $(i, j) \in E$ can be blocked using b_{ij} robots. If we sweep a node $i \in N$ in one step, we must use robots for sweeping i, blocking edges connecting i, and blocking edges connecting all swept and unswept nodes to avoid recontamination. Figure 3 shows an example instance where the numbers on nodes and edges represent the number of robots required for node sweeping and edge blockage, respectively.

The DyPDL model proposed by Kuroiwa and Beck [13] uses a set state variable C to represent swept nodes. A transition τ_i for each node $i \in N$ is defined to add a node i into C and has the precondition $i \notin C$. The number of robots used to sweep i at a state C is $R(i, C) = a_i + \sum_{j \in N} b_{ij} + \sum_{j \in \overline{C} \setminus \{i\}} \sum_{k \in C} b_{jk}$, and an optimal solution minimizes the maximum number of robots used at any step. The optimal value is computed as follows:

$$V(C) = 0, \qquad \text{if } C = N$$
$$V(C) = \min_{i \in \overline{C}} \max\{R(i, C), V(C \cup \{i\})\} \qquad \text{else}$$

We define a transition dominance for the Graph-Clear problem inspired by the customer search model for the Minimization of Open Stacks Problem (MOSP) [4]. In Graph-Clear,

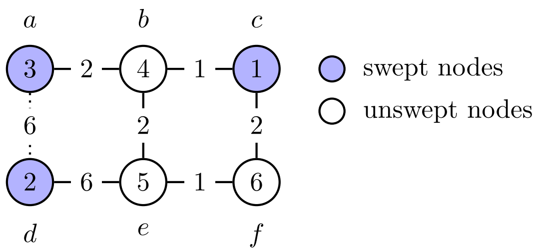


Figure 3 An example instance of Graph-Clear.

observe that in any sequence, a transition τ_i always requires the same number of robots to sweep a node i and block the edges connecting i to its neighbors. The difference arises in the number of robots required to block the *cutting edges* that connect swept nodes in C and unswept nodes in $\overline{C} \setminus \{i\}$. If a transition τ_i reduces the weight of the cutting edges, sweeping any other unswept node i' after applying τ_i requires fewer robots compared to sweeping i' in the current state. Moreover, if τ_i uses fewer robots compared to transition $\tau_{i'}$, then the sequence beginning with transition τ_i then $\tau_{i'}$ uses fewer robots at all steps compared to the corresponding sequence beginning with $\tau_{i'}$.

Consider the instance in Figure 3. Sweeping node b in the next step is preferable to sweeping node e. First, the weight of the cutting edges after sweeping b becomes $(6{+}2{+}2) = 10$, which is smaller than the current state $(2 + 1 + 6 + 2) = 11$. Sweeping e and f afterwards will require fewer robots. Second, sweeping b requires only $4 + (2 + 2 + 1) + (6 + 2) = 17$ robots, which is fewer than the $5 + (2 + 6 + 1) + (2 + 1 + 2) = 19$ robots needed to sweep e. We can conclude that sweeping b then e is always better than sweeping e from the current state. This can be formulated as transition dominance in the model.

▶ **Proposition 14.** *Suppose $i, i' \in \overline{C}$ for a state in the DyPDL model of the Graph-Clear problem. If we have $i \neq i'$ and*

$$a_i + \sum_{j \in \overline{C}} b_{ij} \leq a_{i'} + \sum_{j \in \overline{C}} b_{i'j} \tag{2a}$$

$$\sum_{j \in \overline{C}} b_{ij} \leq \sum_{j \in C} b_{ij} \tag{2b}$$

at a state S, then $\mathit{cost}_{\tau_i}(V(S[\![\tau_i]\!]), S) \leq \mathit{cost}_{\tau_{i'}}(V(S[\![\tau_{i'}]\!]), S)$. ◀

5.3 Discrete Lot Sizing

The discrete lot-sizing problem (DLSP) is a production planning problem for items of various types on a single machine [24]. Each type has a set of items, and the j^{th} item of type i must be produced before its due period $d_{i,j}$. A changeover cost $c_{i,i'}$ is incurred when the machine switches from producing an item of type i to an item of type i'. Additionally, the stocking cost is calculated as the product of the unit cost s_i and the difference between the production period and $d_{i,j}$.

We propose a *sequence model* that makes decisions based on the reversed production sequence of item types. The model uses three types of state variables: an integer variable q represents the latest available period for producing items, t represents the type of item chosen in the last decision, and $\vec{r}$ is a vector where $r_i \in \vec{r}$ indicates the remaining demand for each type i. A transition τ_i represents the decision to produce an item of type i.

Item Parameters

Type	Item No.	$d_{i,j}$	s_i
1	1	3	2
	2	4	2
	3	6	2
	4	9	2
2	1	7	3
	2	9	3

Changeover Costs

	Type 1	Type 2
Type 1	0	3
Type 2	2	0

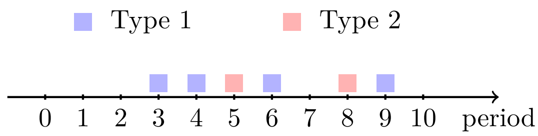


Figure 4 An example instance and solution sequence of DSLP.

Since backlogging is not allowed, the item is produced at the period $\min(d_{i,r_i}, q)$, and the available period is updated to $\min(d_{i,r_i}, q) - 1$. The total cost of the transition is defined as $c(S, i) = c_{i,t} + s_i \cdot (d_{i,r_i} - q)^+$.

Figure 4 illustrates an example instance and a corresponding solution sequence $\sigma = \langle \tau_1, \tau_2, \tau_1, \tau_2, \tau_1, \tau_1 \rangle$. Initially, $q = 10$ and $\vec{r} = \langle 4, 2 \rangle$. The first transition τ_1 selects an item of type 1 to produce at period 9, which is the latest due period for type 1 items. The state variables are updated as follows: q becomes $\min(d_{1,4}, 10) - 1 = 8$, t is updated to 1, and $\vec{r}$ is updated to $\langle 3, 2 \rangle$. The second transition τ_2 selects an item of type 2 to produce at period 8, incurring a stocking cost of $s_2 \cdot (d_{2,r_2} - q)^+ = 3 \cdot (9 - 8) = 3$ and a changeover cost of $c_{2,1} = 2$. The sequence continues until either all items are produced or the sequence becomes invalid, as the available production period q becomes less than the total number of remaining items.

Formally, the DP model can be expressed using the following Bellman equation:

$$
\begin{aligned}
V(S) &= 0, && \text{if } \forall i, r_i = 0 \\
V(S) &= \infty, && \text{if } \textstyle\sum_i r_i > q \\
V(S) &= \min_{\tau_i \in \mathcal{T}(S)} \{c(S, i) + V(S[\![\tau_i]\!])\}, && \text{otherwise}
\end{aligned}
$$

Observe that in the example sequence in Figure 4, the machine is idle at period 7. An item of type 2 could have been produced at this period to reduce the stocking cost. Additionally, this would save the changeover cost incurred from switching from a type 2 item to a type 1 item at period 5. We could verify whether τ_2 dominates τ_1 after applying the prefix transitions $\langle \tau_1, \tau_2 \rangle$. If τ_2 indeed dominates τ_1, the example sequence can be safely discarded, as it is guaranteed not to be optimal. The following proposition formally characterizes the transition dominance observed in this example.

▶ **Proposition 15.** *Suppose the changeover costs satisfy the triangle inequality. If $\tau_i, \tau_{i'} \in \mathcal{T}(S)$ where $i \neq i'$ satisfy:*

- *i has a later production period: $d_{i',r_{i'}} < \min(d_{i,r_i}, q)$,*
- *the total cost is less: $c_{i',i} + c_{i,t} \leq c_{i',t} + 2s_i$,*

at a state S, then $cost_{\tau_i}(V(S[\![\tau_i]\!]), S) \leq cost_{\tau_{i'}}(V(S[\![\tau_{i'}]\!]), S)$. ◀

Note that Coppé, Gillard, and Schaus [5] propose a similar model which introduces a auxiliary idle transition and makes decisions for each period in reverse, starting from the last period

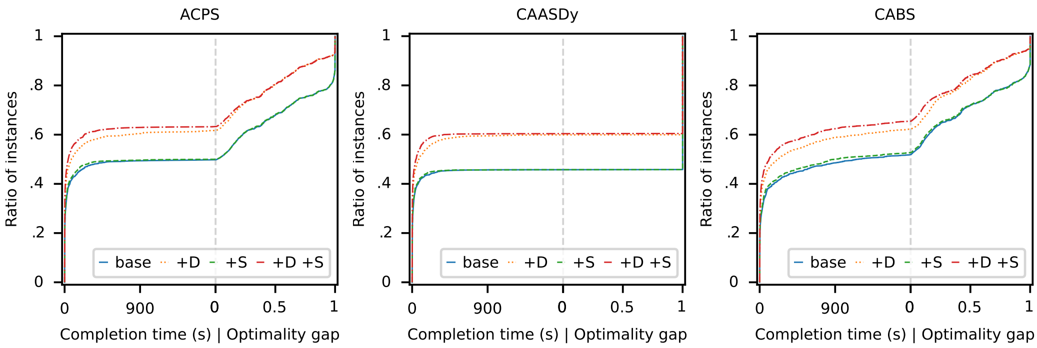


Figure 5 The ratio of instances against time and optimality gap averaged by all problem classes

considered in the planning horizon. Preliminary experiments show that solving the sequence model is more efficient than the period model.

6 Experimental Evaluation

In this section, we experimentally evaluate the impact of using transition dominance and state functions on seven combinatorial optimization problems. For Graph-Clear, OPTW, and TSPTW-M, we modify the existing models[1] by incorporating additional state functions and transition dominance. Following Kuroiwa and Beck [13, 14], we use benchmark instances from the literature. Further details on these instances are provided in Appendix C. Additionally, we introduce new DyPDL models for four problems:

- One Machine Scheduling Minimizing Total Weighted Tardiness ($1|r_i|\sum w_iT_i$): We implement the model based on the one for $1||\sum w_iT_i$ from a public repository,[1] and we generate 300 instances where the number of tasks $n \in \{20, 25, 30, 35, 40\}$, $\alpha \in \{0, 0.5, 1, 1.5\}$ and $\beta \in \{0.05, 0.25, 0.5\}$ control the distribution of release and due dates of tasks. We implement transition dominance following Akturk and Ozdemir [1].
- Aircraft Landing (ALP): transition dominance is implemented based on the DP presented by Lieder, Briskorn and Stolletz [18], and 720 instances from Coppé et al. [6] are used.
- Talent Scheduling (Talent-Sched): the double-ended DP model and the dominance from Qin et al. [25] are implemented in DyPDL, and 1000 instances are sampled from those by Garcia de la Banda and Stuckey [7] in the same way as Kuroiwa and Beck [13].
- Discrete Lot Sizing Problem (DSLP): we implement the sequence model and generate 360 instances following Coppé et al. [5] with the number of items in $\{4, 6, 8, 10\}$, the number of periods in $\{100, 120, 140, 160, 180, 200\}$, and the density in $\{0.7, 0.8, 0.9\}$. The stocking costs and changeover costs are sampled uniformly in $[10, 50]$ and $[20, 40]$ respectively.

The transition dominance interface and state functions are implemented in didp-rs v0.7.3[2] using Rust 1.78.0, and all models are implemented in Python 3.9.16 using DIDPPy, a Python interface for didp-rs. All experiments are run on an Intel Xeon-Gold 6150 processor with a single thread, an 8 GB memory limit, and a time limit of 1800 seconds.

We experiment with three search algorithms: Anytime Column Progressive Search (ACPS), Complete Anytime Beam Search (CABS). and Cost-Algebraic A* Search (CAASDy). These

[1] `https://github.com/Kurorororo/didp-models`
[2] https://didp.ai/

Problem	Solver	Average Solving Time				Average Expanded Nodes		
		base	+D	+S	+D+S	base	+D	reduction
$1\|r_i\|\sum w_i T_i$	ACPS	23.98	1.99	24.03	**1.97**	1803688.77	183502.68	89.83%
	CABS	153.89	3.38	153.46	**3.22**	14889711.97	374543.79	97.48%
	CAASDy	19.98	1.76	20.39	**1.65**	1298236.12	132313.69	89.81%
ALP	ACPS	69.90	47.71	69.55	**43.44**	5290237.80	3154905.57	40.36%
	CABS	165.68	106.90	159.41	**92.69**	14235715.54	8914310.72	37.38%
	CAASDy	67.66	49.18	68.56	**44.61**	3951671.66	2503272.76	36.65%
Graph-Clear	ACPS	34.25	142.44	**26.17**	30.13	1245620.20	947797.81	23.91%
	CABS	40.12	168.04	**24.04**	28.94	1734871.17	952846.14	45.08%
	CAASDy	8.36	17.51	7.11	**4.35**	225567.74	138467.09	38.61%
Talent-Sched	ACPS	191.24	102.09	116.47	**15.11**	1894764.81	394158.89	79.20%
	CABS	238.81	150.64	142.33	**26.35**	2387816.79	665926.33	72.11%
	CAASDy	16.47	13.62	11.28	**2.83**	205820.52	77316.28	62.44%
OPTW	ACPS	57.08	43.49	42.77	**34.74**	1434287.87	1234402.62	13.94%
	CABS	188.23	146.80	143.97	**114.26**	3877856.20	3352288.70	13.55%
	CAASDy	47.48	40.79	35.98	**31.29**	1365988.53	1175550.17	13.94%
TSPTW-M	ACPS	31.97	29.93	15.23	**14.76**	673951.74	619372.42	8.10%
	CABS	98.45	91.26	53.30	**50.64**	2013448.95	1804993.92	10.35%
	CAASDy	30.80	28.19	15.62	**14.53**	623198.30	557875.92	10.48%
DSLP	ACPS	37.72	13.57	37.59	**12.00**	5664688.02	1142853.46	79.82%
	CABS	249.48	105.06	255.71	**93.90**	27283103.00	6743668.77	75.28%
	CAASDy	27.02	10.62	27.33	**9.38**	3313961.61	783309.53	76.36%

Table 1 Experimental results for co-solved instances

solvers usually solve the most instances subject to the limits and exhibit representative tendencies [13] in the DIDP framework. We compare four models for each problem: the "base" model and the base model plus transition dominance "+D", state functions "+S", and both transition dominance and state functions "+D+S".

We compare four configurations of each solving algorithms based on two metrics: (1) *coverage*, which is the number of instances for which optimality or infeasibility is proven within time and memory limits, and (2) *optimality gap*, which is the relative difference between the primal and dual bounds, with a value between 0 and 1. If no solution is found and the instance is not proven infeasible, we set the optimality gap to be 1. Figure 5 illustrates the cumulative ratio of solved instances with respect to completion time and optimality gap. On the left-hand side of each subfigure, the x-axis represents time in seconds, and the y-axis represents the ratio of coverage achieved within x seconds to the total number of instances. On the right-hand side, the x-axis represents the optimality gap, and the y-axis represents the ratio of instances where the optimality gap is less than or equal to x. A curve that is higher and further to the left indicates better performance, which means more instances are solved within a given time or achieve lower optimality gaps within the time limit.

As shown in Figure 5, "+D" and "+D+S" improve the performance of all search algorithms substantially. Upon inspecting the detailed results, combining both transition dominance and state functions enables ACPS, CAASDy, and CABS to solve 420, 450, and 426 more instances in total, respectively, and reduces the average optimality gap by 0.114, 0.155, and 0.106, respectively, within the limits. Compared with ACPS and CABS, the CAASDy solver exhibits an interesting plateau pattern after initial increases, as its first solution is usually optimal: it is guaranteed for our DP models of $1|r_i|\sum w_iT_i$, ALP, Graph-Clear, Talent-Sched, and DSLP in theory, and it is usually the case with OPTW and TSPTW-M in practice. The optimality gap remains 1 even though the dual bound is improved. Overall, the gaps between "+D+S" and "base" show that transition dominance and state functions enable the algorithms to solve instances completely with less time and reduce the primal-dual gaps for unsolved instances considerably.

The coverage and optimality gap may be affected by the time-out and memory limits used in our experiments. To better analyze speed-ups, we also report the average solving time and the average number of expanded nodes in Table 1 for *co-solved instances* that can be solved by all four configurations of each solver. Since the number of expanded nodes remains unchanged when state functions are used, we omit the results for "+S" and "+D+S" in this metric. The fastest average time for each solver is highlighted, and the percentage reduction in expanded nodes is provided for reference.

From the results, we observe that using transition dominance alone ("+D") reduces the number of expanded nodes across all problems and decreases solving time for all problems except Graph-Clear. The "+S" configuration is particularly beneficial in Talent-Sched and OPTW, where state functions help avoid recomputations in their original formulations. Combining both transition dominance and state functions further reduces the average solving time compared to the "base" setup. Overall, the average speed-ups across all co-solved instances for ACPS, CABS, and CAASDy are 10.10, 8.87, and 5.81, respectively.

In the Graph-Clear problem, evaluating transition dominance involves costly summation computations over sets, which significantly slows down performance when applied alone. A closer examination reveals that transition dominance is more effective in reducing expanded nodes for low-density graphs. Combining state functions with transition dominance is essential to achieve better results. In contrast, the "+S" configuration results in similar or higher average times than "base" in $1|r_i|\sum w_iT_i$, ALP, and DSLP, as state functions mostly involve simple integer or Boolean expressions that do not benefit much from caching. They become slightly more effective in "+D+S" when evaluating transition dominance requires more reuses of the cached values. Exploring the trade-offs in using transition dominance and state functions in different problem domains is an interesting direction for future research.

7 Concluding Remarks

In this paper, we define *transition dominance* within the framework of DIDP and introduce new constructs, the *transition dominance interface* and *state function*, into DyPDL to facilitate effective and efficient modelling of transition dominance. We also demonstrate the broad applicability of transition dominance by presenting several previously unexploited cases and show that the insights gained from transition dominance can be applied across problem domains with common combinatorial substructures. Our experimental results indicate that incorporating transition dominance enhances the solving process of various search algorithms in DIDP, reducing computational time and improving solution quality across a range of combinatorial optimization problems.

An interesting research direction is to study whether transition dominance and state functions can be extracted automatically by analyzing DyPDL models. In the benchmark problems studied in this paper, the dominance rules follow recurring patterns. Specifically, transition dominance applies when one can construct a better solution, and the way we construct such a better solution is usually by advancing (shifting forward) a transition. The goal is to identify sufficient conditions under which the objective value of the new solution improves. These construction patterns could serve as a basis for a general method to infer automatically transition dominance from a problem model. Previous work has explored automatic generation of dominance-breaking constraints from constraint programming models [16, 17], and similar techniques may be applicable to analyzing DyPDL models to derive state dominance and transition dominance.

State functions capture common subexpressions and can be extracted through the analysis of DyPDL models, but there is a tradeoff between recomputation and caching. Retrieving results from cache can prevent the recomputation of computationally costly expressions. However, for simple expressions such as basic comparisons, recomputation is more efficient than caching in terms of time and space. Analyzing the trade-off is important for extracting state functions automatically.

References

1. M Selim Akturk and Deniz Ozdemir. An exact approach to minimizing total weighted tardiness with release dates. *IIE Transactions*, 32:1091–1101, 2000. doi:10.1023/A:1013741325877.
2. Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.
3. TC Cheng, JE Diamond, and BM Lin. Optimal scheduling in film production to minimize talent hold cost. *Journal of Optimization Theory and Applications*, 79(3):479–492, 1993. doi:10.1007/BF00940554.
4. Geoffrey Chu and Peter J. Stuckey. Minimizing the maximum number of open stacks by customer search. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 2009. doi:10.1007/978-3-642-04244-7_21.
5. Vianney Coppé, Xavier Gillard, and Pierre Schaus. Decision diagram-based branch-and-bound with caching for dominance and suboptimality detection. *INFORMS Journal on Computing*, 36(6):1522–1542, 2024. doi:10.1287/IJOC.2022.0340.
6. Vianney Coppé, Xavier Gillard, and Pierre Schaus. Modeling and exploiting dominance rules for discrete optimization with decision diagrams. In Bistra Dilkina, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research, CPAIOR 2024*, volume 14742 of *Lecture Notes in Computer Science*, pages 226–242. Springer, 2024. doi:10.1007/978-3-031-60597-0_15.
7. Maria Garcia de la Banda, Peter J. Stuckey, and Geoffrey Chu. Solving talent scheduling with dynamic programming. *INFORMS Journal of Computing*, 23(1):120–137, 2011. doi:10.1287/IJOC.1090.0378.
8. Yvan Dumas, Jacques Desrosiers, Éric Gélinas, and Marius M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2):367–371, 1995. doi:10.1287/OPRE.43.2.367.
9. Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–335, 1998. doi:10.1287/OPRE.46.3.330.
10. Xavier Gillard, Pierre Schaus, and Vianney Coppé. Ddo, a generic and efficient framework for mdd-based optimization. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 5243–5245. International Joint Conferences on Artificial Intelligence Organization, 2020. doi:10.24963/IJCAI.2020/757.
11. Marisa G Kantor and Moshe B Rosenwein. The orienteering problem with time windows. *Journal of the Operational Research Society*, 43(6):629–635, 1992. doi:10.1057/jors.1992.88.
12. Andreas Kolling and Stefano Carpin. The GRAPH-CLEAR problem: definition, theoretical properties and its connections to multirobot aided surveillance. In *2007 IEEE/RSJ*

International Conference on Intelligent Robots and Systems, pages 1003–1008. IEEE, 2007. doi:10.1109/IROS.2007.4399368.

13 Ryo Kuroiwa and J Christopher Beck. Domain-independent dynamic programming: Generic state space search for combinatorial optimization. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 236–244, 2023. doi:10.1609/ICAPS.V33I1.27200.

14 Ryo Kuroiwa and J Christopher Beck. Solving domain-independent dynamic programming problems with anytime heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 245–253, 2023. doi:10.1609/ICAPS.V33I1.27201.

15 Ryo Kuroiwa and J Christopher Beck. Domain-independent dynamic programming. *arXiv preprint arXiv:2401.13883*, 2024. doi:10.48550/arXiv.2401.13883.

16 Jimmy H.M. Lee and Allen Z Zhong. Automatic generation of dominance breaking nogoods for a class of constraint optimization problems. *Artificial Intelligence*, 323:103974, 2023. doi:10.1016/j.artint.2023.103974.

17 Jimmy H.M. Lee and Allen Z Zhong. Exploiting functional constraints in automatic dominance breaking for constraint optimization. *Journal of Artificial Intelligence Research*, 78:1–35, 2023. doi:10.1613/jair.1.14714.

18 Alexander Lieder, Dirk Briskorn, and Raik Stolletz. A dynamic programming approach for the aircraft landing problem with aircraft classes. *European Journal of Operational Research*, 243(1):61–69, 2015. doi:10.1016/j.ejor.2014.11.027.

19 Manuel López-Ibáñez, Christian Blum, Jeffrey W Ohlmann, and Barrett W Thomas. The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. *Applied Soft Computing*, 13(9):3806–3815, 2013. doi:10.1016/j.asoc.2013.05.009.

20 Laurent Michel and Willem-Jan van Hoeve. CODD: A decision diagram-based solver for combinatorial optimization. In Ulle Endriss, Francisco S. Melo, Kerstin Bach, Alberto José Bugarín Diz, Jose Maria Alonso-Moral, Senén Barro, and Fredrik Heintz, editors, *ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 4240–4247. IOS Press, 2024. doi:10.3233/FAIA240997.

21 Roberto Montemanni, Gambardella Luca Maria, et al. An ant colony system for team orienteering problems with time windows. *arXiv preprint arXiv:2305.07305*, 2009. doi:10.48550/arXiv.2305.07305.

22 David R Morrison, Sheldon H Jacobson, Jason J Sauppe, and Edward C Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016. doi:10.1016/j.disopt.2016.01.005.

23 Jeffrey W Ohlmann and Barrett W Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 19(1):80–90, 2007. doi:10.1287/ijoc.1050.0145.

24 Yves Pochet and Laurence A Wolsey. *Production planning by mixed integer programming*, volume 149. Springer, 2006. doi:10.1007/0-387-33477-7.

25 Hu Qin, Zizhen Zhang, Andrew Lim, and Xiaocong Liang. An enhanced branch-and-bound algorithm for the talent scheduling problem. *European Journal of Operational Research*, 250(2):412–426, 2016. doi:10.1016/j.ejor.2015.10.002.

26 Giovanni Righini and Matteo Salani. Dynamic programming for the orienteering problem with time windows. Technical report, Università degli Studi di Milano-Polo Didattico e di Ricerca di Crema, 2006. URL: https://air.unimi.it/handle/2434/6449.

27 Giovanni Righini and Matteo Salani. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & operations research*, 36(4):1191–1203, 2009. doi:10.1016/j.cor.2008.01.003.

28 Sylvie Thiébaux, Jörg Hoffmann, and Bernhard Nebel. In defense of PDDL axioms. *Artificial Intelligence*, 168(1-2):38–69, 2005. doi:10.1016/j.artint.2005.05.004.

29 Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290, 2009. doi:10.1016/j.cor.2009.03.008.

A Proofs of Propositions

Proof for Proposition 3. The proposition is trivial if $V(S^0) = \infty$, as removing transitions in any state does not reduce the cost of that state. Let Ω denote the set of S^0-solutions, which contains a finite number of elements by the assumptions of finiteness of a DyPDL model. We aim to show that there exists an optimal solution such that no transition in the solution is dominated. To achieve this, we define a relation $\mathcal{R}$ over Ω. For any two solutions $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_r \rangle$ and $\sigma' = \langle \sigma'_1, \sigma'_2, \ldots, \sigma'_s \rangle \in \Omega$, we say $\sigma \mathcal{R} \sigma'$ if and only if there exists $t \in \mathbb{Z}_{\geq 0}$ such that: (1) $t \leq r$ and $t \leq s$, (2) $\sigma_i = \sigma'_i$ for all $i \leq t$, and (3) either $\sigma_{t+1} \prec^S \sigma'_{t+1}$, where $S = S^0[\![\sigma_{:t}]\!] = S^0[\![\sigma'_{:t}]\!]$, or $t = r$ and $t < s$. Here, we denote the state resulting from applying $\langle \sigma_1, ..., \sigma_t \rangle$ in S^0 by $S^0[\![\sigma_{:t}]\!]$.

It is straightforward to verify that $\mathcal{R}$ is both transitive and irreflexive. Now, suppose an optimal solution σ^0 for S^0 is pruned due to the restriction of $\mathcal{T}$ to $\mathcal{T}^*$. Then, there must exist another solution σ^1 such that $\sigma^1 \mathcal{R} \sigma^0$. By Definition 2 and Principle of Optimality, $\mathsf{solution\_cost}(\sigma^1, S^0) \leq \mathsf{solution\_cost}(\sigma^0, S^0)$, and σ^1 is also an optimal solution. By repeating this process, we can construct a sequence of optimal solutions $\sigma^0, \sigma^1, \ldots$ such that $\sigma^{i+1} \mathcal{R} \sigma^i$. Since $\mathcal{R}$ is transitive and irreflexive, and the set of optimal solutions is a finite subset of Ω, the sequence cannot repeat indefinitely. The sequence must terminate at some σ^k, which is optimal and not pruned by replacing $\mathcal{T}$ with $\mathcal{T}^*$. ◀

Proof for Proposition 13. We need to show that for any S-solution $\langle \tau_{i',j}; \sigma \rangle$ beginning with $\tau_{i',j}$, there exists a better solution beginning with $\tau_{i,j}$. Suppose $\tau_{i,k} \in \sigma$ where aircraft i lands on runway k. We can always construct another S-solution $\langle \tau_{i,j}, \tau_{i',j}; \sigma' \rangle$, where σ' is σ excluding $\tau_{i,k}$. The landing times of all aircraft on runway j will not change since the landing time of the first aircraft i' is still $t_{i',n_{i'}} = \max(\max(\mathrm{sep}_{c_j,i} + l_j, t_{i,n_i}) + \mathrm{sep}_{i,i'}, t_{i',n_{i'}})$. The landing time of all aircraft on runway k cannot be any later since we are removing an aircraft and the minimum separation times satisfy the triangle inequality. ◀

Proof for Proposition 14. For any S-solution $\langle \tau_{i'}; \sigma \rangle$ beginning with $\tau_{i'}$, τ_i must be in σ since $\tau_i \in \mathcal{T}(S)$ and $i \notin C$. We can construct another solution $\langle \tau_i; \tau_{i'}; \sigma' \rangle$, where σ' is σ excluding τ_i. We now prove that the constructed solution uses an equal or lower number of robots at each step. First, the number of robots required to sweep node i does not exceed that required to sweep node i' at state C:

$$\begin{aligned}
R(i, C) &= a_i + \sum_{j \in N} b_{ij} + \sum_{j \in \overline{C} \setminus \{i\}} \sum_{k \in C} b_{jk} \\
&= a_i + \sum_{j \in N} b_{ij} - \sum_{k \in C} b_{ik} + \sum_{j \in \overline{C}} \sum_{k \in C} b_{jk} \\
&= a_i + \sum_{j \in N \setminus C} b_{ij} + \sum_{j \in \overline{C}} \sum_{k \in C} b_{jk} \\
&\leq a_{i'} + \sum_{j \in N \setminus C} b_{i'j} + \sum_{j \in \overline{C}} \sum_{k \in C} b_{jk} \\
&= a_{i'} + \sum_{j \in N} b_{i'j} + \sum_{j \in \overline{C} \setminus \{i'\}} \sum_{k \in C} b_{jk} \\
&= R(i', C)
\end{aligned}$$

The inequality above is from (2a). Also, the number of robots to sweep other nodes does not increase since for any $l \in \overline{C} \setminus \{i\}$ at a state $C' = C \cup \{i\}$ after applying τ_i

$$\begin{aligned}
R(l, C') =& a_l + \sum_{j \in N} b_{lj} + \sum_{j \in \overline{C'} \setminus \{l\}} \sum_{k \in C'} b_{jk} \\
=& a_l + \sum_{j \in N} b_{lj} + \sum_{j \in \overline{C} \setminus \{l\}} \sum_{k \in C} b_{jk} + \sum_{j \in \overline{C'} \setminus \{l\}} b_{ji} - \sum_{k \in C} b_{ik} \\
\leq& a_l + \sum_{j \in N} b_{lj} + \sum_{j \in \overline{C} \setminus \{l\}} \sum_{k \in C} b_{jk} + \sum_{j \in \overline{C} \setminus \{l\}} b_{ji} - \sum_{k \in C} b_{ik} \\
\leq& a_l + \sum_{j \in N} b_{lj} + \sum_{j \in \overline{C} \setminus \{l\}} \sum_{k \in C} b_{jk} \\
=& R(l, C)
\end{aligned}$$

The first inequality is because $C' \supset C$, and the second inequality is due to (2b). ◀

Proof for Proposition 15. Suppose there exists a dominated sequence of the form $\langle \tau_{i'}, \sigma \rangle$. We can always construct a dominating sequence $\langle \tau_i, \tau_{i'}, \sigma' \rangle$, where the first occurrence of τ_i in σ is removed to form σ'. If the dominated sequence is feasible, then the dominating sequence must also be feasible because the due date $d_{i',r_{i'}}$ is less than $\min(d_{i,r_i}, q)$. Postponing the production of the next item of type i and removing τ_i from σ does not cause the production periods of any items to be pushed earlier.

Next, we prove that the total cost of the dominating sequence is less than that of the dominated sequence. We first analyze the difference in changeover costs. In the dominated sequence, an item of type i' is produced, followed by an item of type t, incurring a changeover cost of $c_{i',t}$. If we insert the production of an item of type i between them, the changeover cost becomes $c_{i',i} + c_{i,t}$. The difference is $c_{i',i} + c_{i,t} - c_{i',t}$. Since the changeover costs satisfy the triangle inequality, removing the first τ_i from σ does not increase the changeover costs.

As for the stocking costs, the only difference lies in the cost of the next item of type i. In the dominated sequence, the item is not produced at period $\min(d_{i,r_i}, q)$ or $d_{i',r_{i'}}$. Thus, its production period must be postponed by at least two periods, increasing the total cost by $c_{i',i} + c_{i,t} - c_{i',t} - 2s_i \leq 0$. ◀

B Additional Problem Descriptions

B.1 One Machine Scheduling Minimizing Total Weighted Tardiness

We consider one machine scheduling for a set of jobs N, where each job $i \in N$ has the processing time p_i, the release dates r_i, the deadline d_i, and the weight w_i, all of which are nonnegative. The objective is to schedule all jobs while minimizing the sum of weighted tardiness, i.e. the different between d_i and the completion time of job i.

We formulate a DyPDL model where one job is scheduled at each step. Let F be a set variable representing the set of scheduled jobs, and t be the current time, which are initially an empty set and 0 respectively. The current time t is an integer resource variable where less is preferred. The completion of a job i when it is scheduled after time t is $C(t, i) = \max(r_i, t) + p_i$, and the tardiness is represented as a numeric expression $T(t, i) = \max(0, C_i(t) - d_i)$. The optimal value of a state $S = (F, t)$ is computed as:

$$\begin{aligned}
&V(S) = 0, && \text{if } F = N \\
&V(S) = \min_{i \in \overline{F}} T(t, i) + V(F \setminus \{i\}, C_i(t)) && \text{else}
\end{aligned}$$

We implement two dominance rules proposed by Akturk and Ozdemir [1].

▶ **Proposition 16.** *Suppose $i, i' \in \overline{F}$ for a state in the DyPDL model of the $1|r_i|\sum w_iT_i$ problem. If $\tau_i, \tau_{i'} \in \mathcal{T}(S)$ satisfy (1) $p_i \geq p_{i'}$, (2) $d_i \leq d_{i'}$, (3) $w_i \geq w_{i'}$, and (4) $C(t,i) \leq C(t,i')$ at a state S, then $cost_{\tau_i}(V(S[\![\tau_i]\!]), S) \leq cost_{\tau_{i'}}(V(S[\![\tau_{i'}]\!]), S)$.* ◀

▶ **Proposition 17.** *Suppose $i \in \overline{F}$ for a state in the DyPDL model of the $1|r_i|\sum w_iT_i$ problem. If $r_i \leq t$, and for all $i' \in \overline{F}$, τ_i satisfies (1) $p_i \leq p_{i'}$, (2) $d_i \leq d_{i'}$, (3) $w_i \geq w_{i'}$ at a state S, then $cost_{\tau_i}(V(S[\![\tau_i]\!]), S) \leq cost_{\tau_{i'}}(V(S[\![\tau_{i'}]\!]), S)$ for all transitions $\tau_{i'} \neq \tau_i$.* ◀

Note that the second dominance rule can be implemented using forced transitions in DyPDL, a transition such that all other transitions are not applicable when it is applicable, a special case of transition dominance.

B.2 Talent Scheduling Problem

The talent scheduling problem [3] is to find a sequence of scenes to shoot to minimize the total cost of a film. In this problem, a set of actors A and a set of scenes N are given. In a scene $s \in N$, a set of actors $A_s \subseteq A$ plays for d_s days. For convenience, let $A(S)$ denote the set of all actors in all scenes $s \in S$, i.e. $A(S) = \cup_{s\in S} A_s$. An actor a incurs the cost c_a for each day they are on location. If an actor plays on days i and j, they are on location on days $i, i+1, ..., j$ even if they do not play on day $i+1$ to $j-1$. The objective is to find a sequence of scenes such that the total cost is minimized.

We use the double-ended search model proposed by Garcia de la Banda et al. [7] and implement the dominance proposed by Qin et al. [25]. Let B and E be two set variables representing the scenes at the beginning and at the end of the schedule, which are empty sets initially, and $R = N \setminus (B \cup E)$ be the set of remaining scenes. At each step, a scene s to shoot is selected from R, and τ_s append s to B. There are two types of actors:

- **Type 1**: If a is neither in $A(B) \cap A(E)$ nor in $A(s)$ but is still present on location during the days of shooting scene s. In other words, $a \notin A(B) \cap A(E)$, $a \notin A(s)$, and $a \in A(B) \cap A(R \setminus \{s\})$. Actor a must be paid for the shooting days of scene s.
- **Type 2**: If a is not included in $A(B) \cap A(E)$ but is included in $A(E)$, and scene s is their first involved scene, then actor a must be paid for all shooting days for scenes in $R \setminus \{s\}$.

Let $\mathrm{T}_1(s, B, E) = (A_s \cup (A(B) \cap A(R \setminus \{s\}))) \setminus (A(B) \cap A(E))$ and $\mathrm{T}_2(s, B, E) = (A_s \cap A(E)) \setminus A(B)$. Therefore, The cost per day to shoot s is

$$c(s, B, E) = d_s \times \sum_{a \in \mathrm{T}_1(s,B,E)} c_a + \sum_{s' \in R \setminus \{s\}} d_{s'} \times \sum_{a \in \mathrm{T}_2(s,B,E)} c_a$$

Overall, we have the following DyPDL model.

$$V(B, E) = \begin{cases} 0 & \text{if } B \cup E = N \\ c(s, B, E) + V(E, B \cup \{s\}) & \text{else} \end{cases}$$

We implement the dual bound where the remaining cost must be at least the total cost of actors times the shooting days they must present, i.e.

$$\eta(S) = \sum_{a \in A(R)} c_a \times \sum_{s \in \{s' | a \in A_{s'}\}} d_s$$

We follow Qin et al. [25] to implement the following dominance rule as transition dominance.

▶ **Proposition 18.** *Suppose two scenes $s, s' \in R$ are two unscheduled scenes. Let $o(B, s) = A(B) \cap \overline{A(B)} \cup A_s$ and $o(E, s) = A(E) \cap \overline{A(E)} \cup A_s$. If τ_s and $\tau_{s'}$ satisfy*

- $o(B, s) \supseteq o(B, s') \wedge o(E, s) \subset o(E, s')$, *or*
- $o(B, s) \subset o(B, s') \wedge o(E, s) \subseteq o(E, s')$,

then $cost_{\tau_s}(V(S[\![\tau_s]\!]), S) \leq cost_{\tau_{s'}}(V(S[\![\tau_{s'}]\!]), S)$ at the current state.

B.3 Orienteering Problem with Time Window

The OPTW problem [11] asks for a schedule to visit a set of customers $N = \{1, \ldots, n-1\}$ starting from the depot 0. Visiting customer j from i incurs travel time $c_{i,j} > 0$ while producing the profit $p_i \geq 0$. Each customer i has a service window $[a_i, b_i]$ and can be visited only within the window. The vehicle needs to wait until a_i upon earlier arrival. The objective is to maximize the total profit while returning to the depot before the deadline b_0.

The DyPDL model we implement is similar to the DP model by Righini and Salani [27]. The model uses a set variable U to represent the set of customers to visit, an element variable loc to represent the current location, and a numeric resource variable t to represent the current time. We visit customers one by one using transitions. Customer j can be visited next if it can be visited and the depot can be reached by the deadline after visiting j. Let $c^*_{i,j}$ be the shortest travel time from i to j. Then, the set of customers that can be visited next is $X(U, loc, t) = \{j \in U \mid t + c_{loc,j} \leq b_j \wedge t + c_{loc,j} + c^*_{j,0} \leq b_0\}$. The optimal value of a state can be computed as follows:

$$\begin{aligned} &V(S) = 0, && \text{if } U = \emptyset \\ &V(S) = -\infty && \text{else if } t + c_{loc,0} > b_0 \\ &V(S) = \max_{j \in X(U,loc,t)} p_j + V(S[\![\tau_j]\!]) && \text{else} \end{aligned}$$

In the actual implementation, we also add additional forced transitions to remove nodes from U that cannot be reached without violating the time limit b_0.

Transition dominance in this problem is similar to that of ALP: if customers i and i' can be visited consecutively without reaching $a_{i'}$ starting from the current position and the current time, then taking $\tau_{i'}$ must not be optimal.

▶ **Proposition 19.** *Suppose the travel times satisfy the triangle inequality, and customers $i, i' \in U$ are unvisited. If $\tau_i, \tau_{i'}$ satisfy that visiting i and i' consecutively does not reach the start time of i', i.e., $\max(a_i, t + c_{loc,i}) + c_{i,i'} < a_{i'}$, then $cost_{\tau_i}(V(S[\![\tau_i]\!]), S) \leq cost_{\tau_{i'}}(V(S[\![\tau_{i'}]\!]), S)$.* ◀

Proof. By definition of $X(U, loc, t)$ we can infer that $X(U, loc, t_1) \subseteq X(U, loc, t_2)$ if $t_1 \leq t_2$. Let the current state be $S = (U, loc, t)$. We show that for any S-solution $\langle \tau_{i'}; \sigma \rangle$ starting with $\tau_{i'}$, we can construct a dominating S-solution starting with τ_i then $\tau_{i'}$ which has at least the same profits.

Suppose that σ does not contain $\tau_{i'}$, we can construct an S-solution $\langle \tau_i, \tau_{i'}; \sigma \rangle$. The state after applying transition $\tau_{i'}$ directly and after applying transitions τ_i and $\tau_{i'}$ are:

$$\begin{aligned} &S_1 = (U \setminus \{i'\}, i', \max(a_{i'}, t + c_{loc,i'})) \\ &S_2 = (U \setminus \{i, i'\}, i', \max(a_{i'}, \max(a_i, t + c_{loc,i}) + c_{i,i'}) = (U \setminus \{i, i'\}, i', a_{i'}) \end{aligned}$$

respectively. The simplification of S_2 is due to the condition in the proposition.

Consider the first transition τ_j in σ. Since $j \neq i$, $j \in X(U \setminus \{i'\}, \text{loc}, t)$ implies $j \in X(U \setminus \{i, i'\}, \text{loc}, t)$. After applying transition τ_j to S_1 and S_2, the current time becomes $\max(a_j, \max(a_{i'}, t + c_{\text{loc},i'}) + c_{i',j})$ and $\max(a_j, a_{i'} + c_{i',j})$, respectively, with the latter term still being less than or equal to the former. Inductively, if σ is an S_1-solution, then it must also be a feasible S_2-solution with the same profit. According to the Bellman equation, the S-solution $\langle \tau_{i'}; \sigma \rangle$ has less profit than $\langle \tau_i, \tau_{i'}; \sigma \rangle$ due to the additional transition τ_i.

Now, suppose the S_1-solution does visit customer i, and let σ' be the sequence of transitions obtained by removing τ_i from σ. We claim that σ' is a feasible S_2-solution. The arguments for the applicability of any transition τ_j before τ_i in σ are similar to those above. Skipping τ_i in the solution does not increase the current time due to the triangle inequality of travel times. After transition τ_i, the set of unvisited customers becomes the same. Therefore, if σ is a feasible S_1-solution, then σ' is a feasible S_2-solution. The difference in the objective between σ and σ' is p_i, and the solutions $\langle \tau_{i'}; \sigma \rangle$ and $\langle \tau_i, \tau_{i'}; \sigma \rangle$ have the same objective. ◀

B.4 Travelling Salesman Problem with Time Windows

In the travelling salesperson problem with time windows and makespan objective [19], a set of customers $N = \{0, ..., n-1\}$ is given. A solution is a tour starting from the depot (index 0), visiting each customer exactly once, and returning to the depot. Visiting customer j from i incurs the travel time $c_{i,j} > 0$. In the beginning, $t = 0$. The visit to customer i must be within a time window $[a_i, b_i]$. Upon earlier arrival, waiting until a_i is required. The objective we consider is to minimize the total makespan where the cost of visiting customer j from the current location i with time t is $\max\{c_{i,j}, a_j - t\}$. Let $c^*_{i,j}$ be the shortest travel time from i to j. Similar to OPTW, the model uses a set variable U represents the set of customers to visit, an element variable loc represents the current location, and a numeric resource variable t represents the current time. We visit customers one by one using transitions. For simplicity, let $X(U, loc, t) = \{j \mid t + c^*_{loc,j} \leq b_j\}$ and $d(t, loc, j) = \max\{c_{loc,j}, a_j - t\}$.

The optimal value of a state S can be computed as follows:

$$
\begin{aligned}
&V(S) = c_{loc,0}, && \text{if } U = \emptyset \\
&V(S) = \infty && \text{else if } \exists j \in U, t + c^*_{ij} > b_j \\
&V(S) = \max_{j \in X(U,i,t)} d(t,i,j) \quad + V(S[\![\tau_j]\!]) && \text{otherwise}
\end{aligned}
$$

▶ **Proposition 20.** *Suppose the travel times satisfy the triangle inequality, and customers $i, i' \in U$ are unvisited. If visiting i and i' consecutively does not reach the start time of i', i.e.* $\max(a_i, t + d(t, loc, i)) + c_{i,i'} < a_{i'}$*, then* $cost_{\tau_i}(V(S[\![\tau_i]\!]), S) \leq cost_{\tau_{i'}}(V(S[\![\tau_{i'}]\!]), S)$. ◀

The proof is similar to that of OPTW.

C Experiment Instances

- **Graph-Clear**: We use 135 instances generated by Kuroiwa and Beck [15], where each graph consists of 20, 30, or 40 nodes.
- **OPTW**: We use 144 instances from Righini and Salani [26], Montemanni and Gambardella [21], and Vansteenwegen et al. [29]. The original instances are defined on a geometric plane. However, rounding distances between locations in the literature may lead to violations of the triangle inequality. To correct this, we update the distance between locations i and j to $d_{ik} + d_{kj}$ whenever there exists a location k such that $d_{ij} > d_{ik} + d_{kj}$.
- **TSPTW-M**: For TSPTW, we use 290 instances from Dumas et al. [8], Gendreau et al. [9], and Ohlmann and Thomas [23]. Similar to OPTW, rounding integer travel times may result in violations of the triangle inequality. We apply the same correction to ensure that the inequality holds for all travel times.