Exact Methods for the Travelling Salesperson Problem with Self-Deleting Graphs

Daniel Pekar
Department of Mechanical and Industrial Engineering, University of Toronto, Canada

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

Abstract

Finding the minimal-cost closed loop on a weighted graph where every vertex is visited exactly once is known as the Travelling Salesperson Problem (TSP). In a recently proposed variant, TSP with Self-Deleting graphs (TSP-SD), visiting a vertex i deletes a set of edges in the graph, preventing their subsequent traversal. Due to the dependency between vertex visits and edge deletion, in TSP-SD the feasibility of a cycle depends on the start node. The best performing solution approaches in the literature rely on a simple problem reformulation to find a backward tour where vertex visits add edges rather than delete them. This paper investigates exact model-based approaches, specifically Constraint Programming (CP), Domain-Independent Dynamic Programming (DIDP), and Mixed Integer Linear Programming (MIP) to solve TSP-SD. We show that simple preprocessing can substantially reduce the options for start/end vertex pairs but typically has a limited positive impact on search performance. Our numerical results demonstrate that the difference between the deletion and addition variants is small for CP and MIP but that the reformulation is critical for DIDP performance. Overall, the DIDP addition model is the best of the exact methods on all test instances and outperforms existing heuristic solvers for small and medium-sized instances while trailing in terms of solution quality on larger instances.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Mathematics of computing → Combinatorial optimization

Keywords and phrases Decision Diagrams & Dynamic Programming, Operations Research & Mathematical Optimization, Modelling & Modelling Languages

Digital Object Identifier 10.4230/LIPIcs.CP.2025.19

Supplementary Material *Software (Source Code)*: `https://github.com/uoft-tidel/tsp-sd`
Dataset (Data): `https://tidel.mie.utoronto.ca/external/Pekar_CP2025/extra.php`

Funding This work was supported the Natural Sciences and Engineering Research Council of Canada.

Acknowledgements Computations were performed on the Niagara supercomputer at the SciNet HPC Consortium. SciNet is funded by ISED Canada; the Digital Research Alliance of Canada; Ontario Research Fund:RE; and the University of Toronto.

1 Introduction

The Travelling Salesperson Problem (TSP) is an NP-hard problem that has been extensively studied since it was first formulated by Hamilton in the 19th century [9]. Variations of the TSP with path or time dependencies, which alter the properties of the graph during its traversal, have also been studied with time-dependent TSP or TSP with time windows being the most common [6, 7, 17]. Carmesin et al. [2] introduced TSP with self-deleting graphs (TSP-SD) where subsets of edges are rendered unavailable after corresponding vertices are visited. TSP-SD has applications in mining, where visits correspond to excavation operations that make traversal of some areas unsafe as well as in driving pile foundations with the related problem of TSP with circle placement [11, 15, 21]. We are interested in TSP-SD as a simple example of a state-dependent problem [8, 14], where problem components, such as costs, can change depending on the decisions that have already been made. With the goal of exploring solution approaches to such problems, this paper compares state-based and constraint-based approaches in terms of ease of modelling and problem solving performance.

Carmesin et al. showed that TSP-SD could be solved by a depth-first search (DFS) that constructs the sequence in reverse order by iteratively adding vertices to the beginning of the tour; vertex visits, therefore, add edges to the graph rather than deleting them. For the DFS, a metaheuristic seeded by a DFS solution, and a later GRASP approach, this backward/addition approach was shown to perform significantly better than searching for forward sequences where visits delete edges [2, 21].

We approach TSP-SD using exact solvers in a model-and-solve paradigm. To that end, eight distinct models are created: addition and deletion variants for two constraint programming (CP) models, one domain-independent dynamic programming (DIDP) model, and one mixed integer programming (MIP) model, respectively. We further observe that the final edge in a tour (i.e., the one that closes the loop from the last vertex visited to the first) must be an edge that is not deleted by any visit. As a consequence, preprocessing can identify a restricted set of first/last pairs that can be easily specified and experimented with in model-based approaches.

Using problem instances from the literature [2], we compare our eight models, with and without first/last vertex restrictions, with existing approaches. We demonstrate that by a significant margin, the DIDP addition model performs best among all the exact approaches and outperforms previous heuristic approaches on small and medium-sized instances while trailing the previous work on larger problems. The addition variants typically exhibit small reductions in measures of search effort compared to the deletion variants with the exception of DIDP where the benefit of the addition model is substantial. The first/last vertex restrictions generally improve performance for finding an initial solution and reducing primal and optimality gaps. However, the addition variants in one CP model and the MIP model performed better without the imposition of first/last vertex restrictions.

This paper is structured as follows. In Section 2, we formally define the problem and summarize related work. Section 3 observes that the first and last vertex pairs in the tour can be restricted through simple preprocessing. In Section 4, we present four exact models each with addition and deletion variants: a CP model based on the ranks of visits, a CP model based on a scheduling perspective, a DIDP model, and a MIP model. Section 5 details the experimentation and our analysis of the results. Finally, Section 6 concludes.

2 Background

We first introduce and define the TSP-SD to give a clearer picture of how it relates to other problems explored in the literature.

2.1 Problem Definition

Given a complete, undirected graph, $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, the TSP with self-deleting graphs (TSP-SD) problem is to find a Hamiltonian tour of minimum length such that no edge is traversed after a visit to a vertex that deletes it [2]. Formally, let c_{ij} be the length of the edge between vertices i and j and let $\mathbf{D}_i$ be the set of edges that are deleted when vertex i is visited. If z_{ij} is a binary variable equal to 1 if edge $\{i, j\}$ is traversed in the tour, then the objective to minimize $\sum_{i,j \in V} c_{ij} z_{ij}$ subject to constraints that ensure a Hamiltonian tour and that edges are not deleted before their use. Carmesin et al. [2] show that TSP-SD is NP-hard. Note

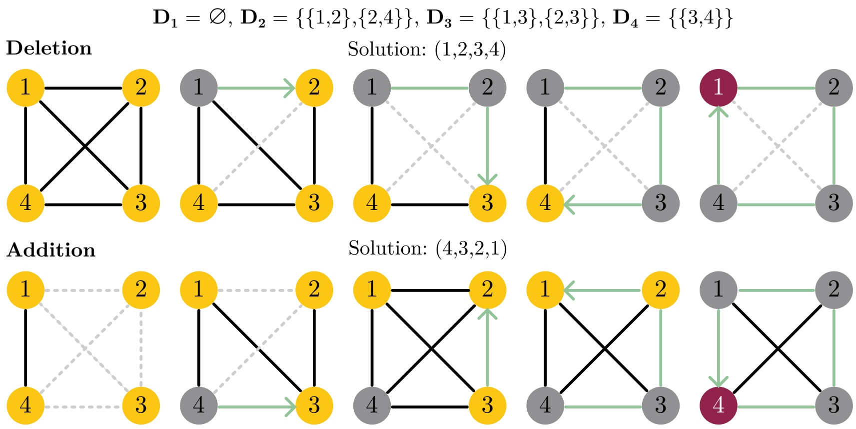

Figure 1 An example of deletion and addition variants of TSP-SD.

that an edge may or may not be incident to a vertex that deletes it, a given edge may be deleted by more than one vertex visit, and that, unlike standard TSP, the vertex at which the tour starts can affect the tour's feasibility. We call this definition of the problem the *deletion* variant.

The top row of Figure 1 shows a four-vertex example. Starting from the complete graph, vertex 1 is chosen as the start node and no edges are deleted as $\mathbf{D}_1 = \emptyset$. Then, edge $\{1, 2\}$ is traversed to vertex 2 and the edges in $\mathbf{D}_2$ are deleted. The deletion of $\{1, 2\}$ has no effect on the tour as that edge has already been traversed. However, now edge $\{2, 4\}$ can no longer be traversed. The sequence continues to vertices 3 and 4 with corresponding deletions of edges in $\mathbf{D}_3$ and $\mathbf{D}_4$ before edge $\{4, 1\}$ is used to return to the start vertex. Of course, finding such a solution without backtracking is not guaranteed.

2.2 Literature Review

Carmesin et al. [2] proposed a depth-first search (DFS) algorithm to solve the TSP-SD. The algorithm begins by choosing a start vertex and then builds the vertex sequence by choosing the next one to visit, while ensuring that the intervening edge has not yet been deleted. When the final vertex is selected, the algorithm ensures that the edge connecting it with the start vertex has not been deleted to ensure that the tour can be completed.

Carmesin et al. also defined a DFS algorithm that constructs the sequence backward from the final vertex to the start vertex. Starting with a graph only containing edges that are not deleted by any vertex visit (i.e., edges $\{j, k\}$ s.t. $\nexists i, \{j, k\} \in \mathbf{D}_i$), a vertex is inserted at the beginning of the partial sequence, while ensuring that the intervening edge exists. Under this regime, a visit to vertex i *adds* the edges $\mathbf{D}_i$ to the graph. If an edge appears in multiple deletion sets, then all corresponding vertex visits must be present in the partial sequence before the edge can be used to add a vertex to the beginning of the partial sequence. When the starting vertex is selected (i.e., in the last step of the DFS), it is not sufficient for the edge connecting it to the final node to be present in the current graph because in the forward sequence the edge connecting the first and last vertices is the last edge to be traversed. Thus, to close the tour, the last edge must exist after all vertices have been visited (i.e., it must be an edge that no vertex deletes). The DFS algorithm performs this extra check when adding the starting vertex and backtracks if it is not satisfied. Carmesin et al. prove the correctness of their backward DFS. We refer to this approach to solving as the *addition* variant.

The second row of Figure 1 illustrates the backward DFS algorithm. Vertex 4 is selected as the last vertex and the edges in $\mathbf{D}_4$ are added to the graph. Vertices 3, 2, and 1 are then iteratively added to the beginning of the sequence traversing edges that exist at the time of the traversal. As noted, the DFS checks that the edge $\{1, 4\}$ existed in the starting graph to ensure that the forward tour can be completed.

Carmesin et al. show, experimentally, that the backward DFS outperforms the forward DFS in finding feasible solutions and proving infeasibility, requiring significantly fewer search nodes and less time. Further work [2, 21], proposed a metaheuristic, warm-started with a solution found by a time-limited backward DFS, and a Greedy Randomized Adaptive Search Procedure (GRASP) [5]. The metaheuristic improved on the backward DFS and GRASP further improved performance, finding best and mean solutions up to 16% and 6.7% better, respectively, than the warm-started metaheuristic on large instances.

While TSP-SD shares similarities with other time- or path-dependent routing problems such as the time-dependent TSP (TDTSP) [18], the closest work is by Lipovetzky et al. [15]. That work investigates AI planning for a mining application where parts of the mine are only physically accessible after previous blocks have been excavated. These physical considerations impose partial orderings of operations, similar to TSP-SD, but whereas the mining operations add subsequent travel paths, vertex visits in TSP-SD remove them.

3 First/Last Vertex Constraints

Before presenting our exact models of TSP-SD, we make the following observation.

▶ **Observation 1.** *Since all vertices are visited upon completion of a Hamiltonian path, in any solution, all edges that could be deleted are deleted. As such, the set of undeleted edges at the end of the tour,* $\boldsymbol{E}_{remain}$*, can be defined as:* $\boldsymbol{E}_{remain} = \boldsymbol{E} \setminus \bigcup_{i=1}^{n} \boldsymbol{D}_i$.

Since the tour must return to the start vertex, the returning edge must be an edge in $\boldsymbol{E}_{remain}$*, and therefore the start and end vertices of the tour must be incident to the remaining edges. From this we can define* $\boldsymbol{V}_{remain}$ *as:* $\boldsymbol{V}_{remain} = \bigcup\{i, j\}, \forall\{i, j\} \in E_{remain}$.

While the existing DFS approaches correctly ensure that the edge between the first and last vertices in the tour is not deleted, they do not exploit this observation. No restrictions are made on the choice of the start vertex nor is any reasoning done about partial tours with remaining return edges to the first vertex. We implement these first/last restrictions in our models and evaluate their impact on problem solving.

4 TSP-SD Models

Given the performance differences in the literature between the deletion and addition variants and their embedding in a DFS search, we are interested in understanding if such differences are manifest in model-based approaches. Thus, we formulated four distinct models (two CP models, one DIDP model, and one MIP model) each with a deletion and addition variant. Further, we investigate Observation 1 by testing each of the eight models with and without first/last restrictions. The notation we adopt is summarized in Table 1.

4.1 Constraint Programming

We present two CP models: a rank-based model, where the main decision variable is the position of each vertex in the tour, and a scheduling-based model, where we use optional interval variables to represent the sequence of edge traversals.

Table 1 Notation used to define TSP-SD.

Symbol	Explanation
c_{ij}	Distance from vertex i to vertex j
$\mathbf{V}$	Set of all vertices $\{1, \ldots, n\}$
$\mathbf{E}$	Set of all edges
$\mathbf{D}_i$	Set of edges deleted after visiting vertex i
$\mathbf{D}_{ij}$	Set of vertices which delete the edge $\{i, j\}$
$\mathbf{E}_{remain}$	Set of remaining edges after all vertices are visited
$\mathbf{V}_{remain}$	Set of vertices along remaining edges

4.1.1 CP Rank Model

The CP Rank model is premised on the idea that each vertex must be assigned a unique rank (i.e., position in the sequence) and each rank must be associated with a unique vertex. Let $x_i \in X$ be the vertex visited at rank i in the tour and let $y_j \in Y$ be the rank of vertex j in the tour. The domains of both variables are $\{1, ..., n\}$ for n vertices.

The *CP Rank Del* model is defined in Figure 2. The objective function seeks to minimize the sum of costs between vertices in consecutive ranks, with the modulo function accounting for the return edge. Constraint (1b) ensures that $x_{y_i} = y_{x_i}$. Each vertex is visited exactly once so the values of both X and Y must form permutations as enforced with constraints (1c) and (1d). Constraint (1e) expresses the deletion behavior by constraining the ordering of vertex visits and edge traversals. For every deleted edge, $\{j, k\} \in \mathbf{D}_i$, either the two vertices j, k are not adjacent in the sequence or visits to both vertices j and k must be before the visit to i. Note that the ranks of vertex i and either vertex j or k may be equal as it is possible to traverse an edge to a vertex that deletes it. That is, i may be equal to j or k. Constraint (1f) ensures that the edge between the first and last vertices is not deleted by any vertex visit.

We can modify CP Rank Del to represent the addition variant of the problem simply by reversing the ordering in constraint (1e): an edge $\{j, k\}$ can be traversed only after visiting every vertex i in the set $\mathbf{D}_{jk} = \{i : \{j, k\} \in \mathbf{D}_i\}$. Thus, constraint (1e) is replaced with constraint (3).

$$\min \sum_{i \in V} c_{i x_{(y_i \bmod n)+1}} \tag{1a}$$

$$\text{s.t. } \textsc{Inverse}(X, Y) \tag{1b}$$

$$\textsc{AllDifferent}(X) \tag{1c}$$

$$\textsc{AllDifferent}(Y) \tag{1d}$$

$$(|y_j - y_k| \neq 1) \vee (y_j \leq y_i \wedge y_k \leq y_i) \qquad \forall \{j, k\} \in \mathbf{D}_i \quad \forall i \in \mathbf{V} \tag{1e}$$

$$|y_j - y_k| \neq n - 1 \qquad \forall \{j, k\} \in \mathbf{D}_i \quad \forall i \in \mathbf{V} \tag{1f}$$

$$\text{Integer variable} \quad x_i = \{1, \ldots, n\} \qquad \forall i \in \mathbf{V} \tag{1g}$$

$$\text{Integer variable} \quad y_j = \{1, \ldots, n\} \qquad \forall i \in \mathbf{V} \tag{1h}$$

Figure 2 The CP Rank Del model.

$$\min \sum_{i \in V} c_{i x_{(y_i \bmod n)+1}} \tag{2a}$$

$$\text{s.t. } (1b), (1c), (1d), (1e)$$

$$\textsc{AllowedAssignments}(\{x_1, x_n\}, \{\{i,j\} \in \mathbf{E}_{remain}\}) \tag{2b}$$

$$\text{Integer variable} \quad x_i = \begin{cases} \{1, \dots, n\} & \forall \quad i \in \{2, \dots, n-1\} \\ \mathbf{V}_{remain} & \forall \quad i \in \{1, n\} \end{cases} \tag{2c}$$

$$\text{Integer variable} \quad y_j = \begin{cases} \{2, \dots, n-1\} & \forall \quad j \notin \mathbf{V}_{remain} \\ \{1, \dots, n\} & \forall \quad j \in \mathbf{V}_{remain} \end{cases} \tag{2d}$$

Figure 3 The CP Rank Del model with first/last vertex restrictions.

$$(|y_j - y_k| \neq 1) \vee (y_j \geq y_i \wedge y_k \geq y_i) \quad \forall \{j,k\} \in \mathbf{D}_i \quad \forall i \in \mathbf{V} \tag{3}$$

We call this second formulation *CP Rank Add*. The solutions to CP Rank Add is a solution to CP Rank Del with each x_i, y_j in the addition variant being equal to x_{n-i+1}, y_{n-i+1} in the deletion variant.

We can include Observation 1 in the CP Rank models by directly constraining the domains of the corresponding variables in X and Y as in the model presented in Figure 3. We replace constraint (1f) with a table constraint specifying the possible return edges and restrict the domains of the variables x_i and y_j in constraints (2c) and (2d), respectively.

4.1.2 CP Interval Model

Motivated by the success of CP scheduling constructs both in scheduling and non-scheduling problems [10, 16], we present the intuition of the CP Interval model in Figure 4. In the model, each edge $\{i,j\}$ is represented as an optional interval variable, $traverse_{ij}$, with length c_{ij}. As each vertex i must have exactly one edge entering and one edge exiting it in a tour, we create two interval variables, in_i and out_i, and employ an ALTERNATIVE constraint to ensure that in_i is set to the chosen $traverse_{ji}$ variable that enters vertex i and out_i is set to the chosen $traverse_{ij}$ variable that exits vertex i. Each $traverse_{ij}$ variable appears in two

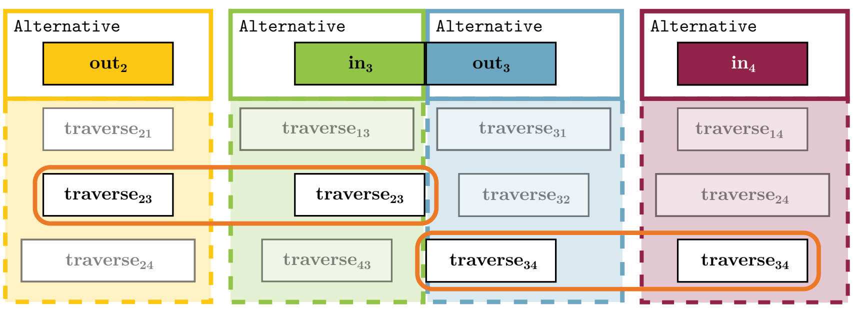

Figure 4 An illustration of the interval variables in the CP Interval model. The traversal of an edge $\{i,j\}$ is represented by an optional interval variable $traverse_{ij}$ that occurs in the ALTERNATIVE constraints with the out_i and in_j interval variables.

$$
\begin{array}{llr}
& \min \textsc{EndOf}(in_{n+1}) & \text{(4a)} \\
\text{s.t.} & \textsc{NoOverlap}(\text{INS}) & \text{(4b)} \\
& \textsc{Last}(\text{INS}, in_{n+1}) & \text{(4c)} \\
& \textsc{StartAtEnd}(out_i, in_i) \qquad \forall i \in \mathbf{V} & \text{(4d)} \\
& \textsc{Alternative}(in_i, traverse_{ji} \forall j \in \mathbf{V}) \qquad \forall i \in \mathbf{V}) & \text{(4e)} \\
& \textsc{Alternative}(out_i, traverse_{ij} \cup traverse\_last_{ij} \forall j \in \mathbf{V}) \qquad \forall i \in \mathbf{V} \setminus \{0\}) & \text{(4f)} \\
& \textsc{Alternative}(in_{n+1}, traverse\_last_{ij}) \qquad \forall \{i,j\} \in \mathbf{E} & \text{(4g)} \\
& \textsc{IsPresent}(traverse\_last_{ij}) \leftrightarrow & \\
& \quad (\textsc{IsPresent}(traverse_{0i}) \land \textsc{IsPresent}(traverse_{j,n+1})) \qquad \forall \{i,j\} \in \mathbf{E} & \text{(4h)} \\
& \textsc{EndBeforeEnd}(traverse_{jk}, in_i \quad \forall \{j,k\} \in \mathbf{D}_i) \qquad \forall i \in \mathbf{V} & \text{(4i)} \\
& \neg \textsc{IsPresent}(traverse\_last_{jk}) \qquad \forall \{j,k\} \in \mathbf{D}_i, i \in \mathbf{V} & \text{(4j)} \\
& \text{Optional interval variable} \quad traverse_{ij} \quad size = c_{ij} \qquad \forall \{i,j\} \in \mathbf{E} & \text{(4k)} \\
& \text{Optional interval variable} \quad traverse_{0i} \quad size = 0 \qquad \forall i \in \mathbf{V} & \text{(4l)} \\
& \text{Optional interval variable} \quad traverse\_last_{ij} \quad size = c_{ij} \qquad \forall \{i,j\} \in \mathbf{E} & \text{(4m)} \\
& \text{Interval variable} \quad in_i \qquad \forall i \in \mathbf{V} \cup \{n+1\} & \text{(4n)} \\
& \text{Interval variable} \quad out_i \qquad \forall i \in \mathbf{V} \cup \{0\} & \text{(4o)} \\
& \text{Sequence variable INS} \quad \{in_i \quad \forall i \in \mathbf{V} \cup \{n+1\}\} & \text{(4p)}
\end{array}
$$

Figure 5 The CP Interval Del model.

ALTERNATIVE constraints, one related to out_i and one for in_j, and thus when the solver sets $traverse_{ij}$ as present, out_i will correctly equal in_j. We further constrain out_i to start at the end of in_i as visits are instantaneous. Finally, though not illustrated in Figure 4, the in_i variables are sequenced to form a permutation.

Figure 5 presents the formal definition of *CP Interval Del*. In addition to the interval variables described above, we introduce two dummy vertices, with indices $i = 0$ and $i = n+1$, as the start and end points in the permutation with corresponding interval variables out_0, in_{n+1}, $traverse_{0j}$, and $traverse_{j,n+1}$. The return edges are represented by another interval variable, $traverse\_last_{ij}$, that are alternative realizations of in_{n+1}.

We seek to minimize the end time in_{n+1} in objective (4a). Constraints (4b) and (4c) constrain all in_i variables to form a permutation with in_{n+1} coming last. Constraint (4d) ensures that the in_i and out_i variables are sequenced contiguously. The ALTERNATIVE constraints (4e)-(4g) represent the logic described in Figure 4 extended to include the return edge. Constraint (4h) defines the return edge to be between the vertices visited immediately after the dummy start vertex and immediately before the dummy end vertex. The self-deletion requirement is modeled by sequencing of intervals out_i and intervals $traverse_{jk}$ for $\{j,k\} \in \mathbf{D}_i$. Finally, constraint (4j) prevents a deleted edge from being the return edge.

To define the *CP Interval Add* model, the deletion constraint (4i) is replaced with (5), such that if $traverse_{jk}$ is present, it must occur after the visit to the vertex that adds it.

$$\textsc{StartBeforeStart}(out_i, traverse_{jk}) \quad \forall \{j,k\} \in \mathbf{D}_i \quad \forall i \in \mathbf{V} \tag{5}$$

To incorporate first/last vertex restrictions, we adjust the domains of the $traverse_{0i}$

$$\min \text{EndOf}(in_{n+1}) \quad (6a)$$

s.t.

$$(4b), (4c), (4d), (4e), (4i)$$

$$\text{Alternative}(out_i, traverse_{ij} \cup traverse\_last_{ik} \forall j \in \mathbf{V} \; k \in \mathbf{V}_{remain} : \{i,k\} \in E_{remain}) \quad \forall i \in \mathbf{V} \setminus \{0\}) \quad (6b)$$

$$\text{Alternative}(in_{n+1}, traverse\_last_{ij}) \quad \forall \{i,j\} \in \mathbf{E}_{remain} \quad (6c)$$

$$\text{Alternative}(out_0, traverse_{0i}) \quad \forall i \in \mathbf{V}_{remain} \quad (6d)$$

$$\text{IsPresent}(traverse_{0i}) = \sum_{(j,i) \in E_{remain}} \text{IsPresent}(traverse\_last_{ji}) \quad \forall i \in \mathbf{V}_{remain} \quad (6e)$$

$$\text{Optional interval variable} \quad traverse_{0i} \quad size = 0 \quad \forall i \in \mathbf{V}_{remain} \quad (6f)$$

$$\text{Optional interval variable} \quad traverse\_last_{ij} \quad size = c_{ij} \quad \forall \{i,j\} \in \mathbf{E}_{remain} \quad (6g)$$

Figure 6 The CP Interval Del model with first/last vertex restrictions.

and $traverse\_last_{ij}$ variables in Figure 6. The domain of $traverse_{0i}$ is restricted $\mathbf{V}_{remain}$ (constraint (6f)), while the domain of $traverse\_last_{ij}$ is restricted to $\mathbf{E}_{remain}$ (constraint (6g)). The same domain restrictions are similarly replicated in constraints (6b), (6c), and (6d). We also replace constraint (4h) in the CP Interval Del model with constraint (6e) to link the first traverse variable with the corresponding $traverse\_last_{ji}$ variables.

4.2 Domain-Independent Dynamic Programming

DIDP is a declarative model-based paradigm for combinatorial optimization based on dynamic programming [12]. Our TSP-SD model builds on existing formulations for TSPTW [4, 13] and assembly line balancing problem with sequence-dependent setup times [22]. The deletion variant, *DIDP Del*, is shown in Figure 7.

The state variables are U, the set of unvisited vertices, i, the current vertex, and f, the first vertex visited in the tour. As in the CP Interval model we let vertex 0 represent a dummy start node.

We aim to compute the value function of target state $\langle N \setminus \{0\}, 0, 0 \rangle$ (term (7a)). Equation (7b) recursively computes the cost of the state as we traverse each edge. When $i = 0$, the transition to the start vertex is selected. Subsequently, while the set of unvisited vertices is not a singleton, the next vertex is chosen ensuring that the intervening edge has not been deleted. This condition is expressed as $D_{ij} \subseteq U$, as all vertices that delete edge $\{i, j\}$ must still be unvisited. The selection of the final vertex is a special case, captured in the next condition that requires that the return edge has not been deleted: $D_{jf} \subseteq U$. The base case of $U = \emptyset$ has a cost of 0. If none of these conditions are satisfied, the state is a deadend and so is assigned the cost of ∞. Following the TSPTW model [12], we specify two dual bounds, inequalities (7c) and (7d), based on the minimum cost of all incoming edges from or all outgoing edges to the unvisited vertices.

To create the addition variation, *DIDP Add*, the Bellman equation is replaced by (8). In the second and third cases, rather than ensuring the traversed edge has not yet been deleted, we require that all adding vertices must be already visited: $D_{ij} \cap U = \emptyset$.

$$\text{compute } \mathcal{V}(N \setminus \{0\}, 0, 0) \tag{7a}$$

$$V(U,i,f) = \begin{cases} \min_{j \in V} V(U \setminus \{j\}, j, j) & \text{if } i = 0 \\ \min_{j \in U | D_{ij} \subseteq U} c_{ij} + V(U \setminus \{j\}, j, f) & \text{else if } |U| > 1 \wedge (\exists j \in U \mid D_{ij} \subseteq U) \\ \min_{j \in U} c_{ij} + c_{jf} + V(U \setminus \{j\}, j, f) & \text{else if } |U| = 1 \wedge \\ & \quad (\exists j \in U \mid D_{fj} = \emptyset \wedge D_{ij} \subseteq U) \\ 0 & \text{else if } U = \emptyset \\ \infty & \text{else} \end{cases} \tag{7b}$$

$$V(U,i,f) \geq \sum_{j \in U \setminus \{i\}} \min_{k \in N \setminus \{j\}} c_{kj} \tag{7c}$$

$$V(U,i,f) \geq \sum_{j \in U \setminus \{f\}} \min_{k \in N \setminus \{j\}} c_{jk} \tag{7d}$$

Figure 7 The DIDP Del model.

$$V(U,i,f) = \begin{cases} \min_{j \in V} V(U \setminus \{j\}, j, j) & \text{if } i = 0 \\ \min_{j \in U | D_{ij} \cap U = \emptyset} c_{ij} + V(U \setminus \{j\}, j, f) & \text{else if } |U| > 1 \wedge \\ & \quad (\exists j \in U \mid D_{ij} \cap U = \emptyset) \\ \min_{j \in U} c_{ij} + c_{jf} + V(U \setminus \{j\}, j, f) & \text{else if } |U| = 1 \wedge \\ & \quad (\exists j \in U \mid D_{fj} = \emptyset \wedge D_{ij} \cap U = \emptyset) \\ 0 & \text{else if } U = \emptyset \\ \infty & \text{else} \end{cases} \tag{8}$$

Figure 8 The Bellman equation in the DIDP Add model.

Finally, to incorporate first/last vertex restrictions into the DIDP Del model, the possible choices for the first vertex are restricted to only those in set $\mathbf{V}_{remain}$. Similarly, the possible choices for the last vertex are restricted to only those such that the edge $\{j, f\}$ is in the set of remaining edges $\mathbf{E}_{remain}$. The Bellman equation is shown in Figure 9.

4.3 Mixed Integer Programming

We develop a MIP model similar to the one for Time Dependent TSP [17]. The primary difference, and an advantage in TSP-SD, is that the variables are indexed by rank as opposed to time slot and thus do not scale with the time horizon, only with the number of vertices.

Let binary decision variable x_{ijr} represent the traversal of edge $\{i, j\}$ at rank r. As such, the number of binary decision variables for a given graph is $O(n^3)$. To account for the cost of returning to the starting vertex, we introduce the binary variable y_{ij} that is 1 if $\{i, j\}$ is the return edge.

Figure 10 shows the *MIP Del* model. The objective is to minimize the total distance

$$V(U,i,f) = \begin{cases} \min\limits_{j \in V_{remain}} V(U \setminus \{j\}, j, j) & \text{if } i = 0 \\ \min\limits_{j \in U \mid D_{ij} \subseteq U} c_{ij} + V(U \setminus \{j\}, j, f) & \text{else if } |U| > 1 \land (\exists j \in U \mid D_{ij} \subseteq U) \\ \min\limits_{j \in U} c_{ij} + c_{jf} + V(U \setminus \{j\}, j, f) & \text{else if } |U| = 1 \land \\ & \quad (\exists j \in U \mid (j,f) \in \mathbf{E}_{remain} \land D_{ij} \subseteq U) \\ 0 & \text{else if } U = \emptyset \\ \infty & \text{else} \end{cases} \tag{9}$$

Figure 9 The Bellman equation for the DIDP Del model with first/last vertex restrictions.

traveled on all intermediate edges plus the return edge. Constraints (10b) and (10c) ensure that each vertex is entered once and each rank is chosen once. The proper rank of edges is maintained via constraint (10d). The next three constraints specify that the return edge cannot be deleted (10e), that it must connect vertices in the first and last rank (10f), and that there can be only one return edge (10g). Finally, constraint (10h) ensures that the rank of any deleted edge must be less than or equal to any vertex that deletes it.

For the addition variant, *MIP Add*, constraint (10h) is replaced with constraint (11) to ensure that either an added edge is ranked after all adding vertices or the edge is not used.

$$\sum_{r \in \mathbf{V}} r x_{klr} + M(1 - \sum_{r \in \mathbf{V}} x_{klr}) \geq 1 + \sum_{r \in \mathbf{V}} \sum_{j \in \mathbf{V}} r x_{jir} \quad \forall \{k,l\} \in \mathbf{D}_i \quad \forall i \in V \tag{11}$$

We can adjust the MIP Del model to include first/last vertex restrictions by modifying the constraints that affect the choice of first and last vertex and the scope of variable y_{ij} as shown in Figure 11. Constraints (12b) and (12c) are added to restrict the first and last vertices to only those in the set $\mathbf{V}_{remain}$. We ensure that the first and last vertices are connected by an edge in $\mathbf{E}_{remain}$ using constraint (12d). Finally, constraints (12e) and (12f) are analogous to (10f) and (10g), only restricted to the set of possible edges in $\mathbf{E}_{remain}$.

5 Numerical Experiments

The aims of our numerical experiments are to evaluate the performance of the exact models, to investigate whether the deletion or addition variants perform differently, to assess the impact of the first/last restrictions, and finally to compare the best-performing model-based techniques to the existing custom approaches. We address the first three aims in Experiment 1 and the final one in Experiment 2.[1]

Recall that Carmesin et al. [2] found a substantial computational advantage in adopting the addition variant within a depth-first search that built the sequence backward. For our CP and MIP models, the difference between the addition and deletion variants is simply the direction of a set of sequencing constraints. Further, as the corresponding solvers do

[1] GitHub: `https://github.com/uoft-tidel/tsp-sd`
Instance Data and Results: `https://tidel.mie.utoronto.ca/external/Pekar_CP2025/extra.php`

$$\min \sum_{r\in\mathbf{V}} \sum_{\{i,j\}\in\mathbf{E}} c_{ij}x_{ijr} + \sum_{\{i,j\}\in\mathbf{E}} c_{ij}y_{ij} \tag{10a}$$

$$\text{s.t.} \sum_{i\in\mathbf{V}} \sum_{r\in\mathbf{V}} x_{ijr} = 1 \qquad \forall j \in \mathbf{V} \tag{10b}$$

$$\sum_{\{i,j\}\in\mathbf{E}} x_{ijr} = 1 \qquad \forall r \in \mathbf{V} \tag{10c}$$

$$\sum_{j\in\mathbf{V}} x_{ijr} = \sum_{j\in\mathbf{V}} x_{j,i,r-1} \qquad \forall i \in \mathbf{V}, r \in \mathbf{V} \tag{10d}$$

$$y_{jk} = 0 \qquad \forall \{j,k\} \in \mathbf{D}_i, i \in \mathbf{V} \tag{10e}$$

$$x_{0i0} + \sum_{k\in\mathbf{V}} x_{kjn} \le y_{ij} + 1 \qquad \forall \{i,j\} \in \mathbf{E} \tag{10f}$$

$$\sum_{(i,j)\in\mathbf{E}} y_{ij} = 1 \tag{10g}$$

$$\sum_{r\in\mathbf{V}} r x_{klr} \le \sum_{r\in\mathbf{V}} \sum_{j\in\mathbf{V}} r x_{jir} \qquad \forall \{k,l\} \in \mathbf{D}_i, i \in \mathbf{V} \tag{10h}$$

$$x_{ijr} = \begin{cases} 1 & \text{if traversing edge } \{i,j\} \text{ at rank } r \\ 0 & \text{else} \end{cases} \qquad \forall i,j \in \mathbf{V}, r \in \mathbf{V} \tag{10i}$$

$$y_{ij} = \begin{cases} 1 & \text{if traversing edge } \{i,j\} \text{ last} \\ 0 & \text{else} \end{cases} \qquad \forall \{i,j\} \in \mathbf{E} \tag{10j}$$

Figure 10 The MIP Del model.

not necessarily build the sequence in order (either backward or forward), we expect limited performance differences between the variants. In contrast, the DIDP models are solved by adding vertices to the start or end of a partial sequence. Thus, we expect to see a similar effect as observed in the previous work with the addition variant performing better than deletion. As the first/last restrictions limit the search space, we expect to see their inclusion will improve performance of all of our models.

5.1 Experiment 1: Comparison of Exact Models

Problem Set. The exact models were compared using a dataset of 60 instances chosen from an existing set of 30,000 randomly generated problems with size $n = [10, 20, \ldots, 100, 150, 200]$ and 50 instances per size [20]. For each size, a deletion function was randomly generated with differing probabilities to produce instances with varying expected vertex density over the course of the tour (see Carmesin et al. [2] for the exact definition of density used). For each n, we chose the first instance at each quintile of average vertex degree (0%, 25%, 50%, 75%, 100%), producing 60 instances in total with diverse sizes and densities.

We did not filter the problem instances for feasibility. A posteriori, at least one of our exact methods found a feasible solution or proved infeasibility for each instance showing that the set consists of 39 feasible and 21 infeasible instances.

Experimental Set-up. Each model is run with a single-thread with a 30-minute time-out and an 8 GB memory limit for each instance. All model-based approaches are run on a dedicated Linux server, with Intel(R) Xeon(R) Gold 6148 CPUs running at 2.4 GHz. The

$$\min \sum_{r \in \mathbf{V}} \sum_{\{i,j\} \in \mathbf{E}} c_{ij} x_{ijr} + \sum_{\{i,j\} \in \mathbf{E}_{remain}} c_{ij} y_{ij} \tag{12a}$$

$$\text{s.t.} \quad (10b), (10c), (10d), (10h)$$

$$\sum_{i \in \mathbf{V}_{remain}} x_{0i0} = 1 \tag{12b}$$

$$\sum_{i \in \mathbf{V}} \sum_{j \in \mathbf{V}_{remain}} x_{ijn} = 1 \tag{12c}$$

$$x_{0i0} \leq \sum_{j \in \mathbf{V}_{remain} | (j,i) \in \mathbf{E}_{remain}} \sum_{k \in \mathbf{V}} x_{kjn} \qquad \forall i \in \mathbf{V}_{remain} \tag{12d}$$

$$x_{0i0} + \sum_{k \in \mathbf{V}} x_{kjn} \leq y_{ij} + 1 \qquad \forall \{i,j\} \in \mathbf{E}_{remain} \tag{12e}$$

$$\sum_{\{i,j\} \in \mathbf{E}_{remain}} y_{ij} = 1 \tag{12f}$$

$$x_{ijr} = \begin{cases} 1 & \text{if traversing edge } \{i,j\} \text{ at rank } r \\ 0 & \text{else} \end{cases} \qquad \forall i,j \in \mathbf{V}, r \in \mathbf{V} \tag{12g}$$

$$y_{ij} = \begin{cases} 1 & \text{if traversing edge } \{i,j\} \text{ last} \\ 0 & \text{else} \end{cases} \qquad \forall \{i,j\} \in \mathbf{E}_{remain} \tag{12h}$$

Figure 11 The MIP Del model with first/last vertex restrictions.

CP models were run using CP Optimizer 22.1.1.0 via IBM DOcplex, DIDP was run using the CABS solver via DIDPPy v0.8.0, and the MIP models were solved by Gurobi 12.0.0.

Model Comparison. An overview of the results can be seen in Figures 12 and 13 with detailed results in Appendix A.

The DIDP Add model and both CP Rank models outperform the others, with DIDP Add having the overall advantage. DIDP Add proved optimality or infeasibility on the largest number of instances, produced substantially better solutions, and achieved the best optimality gap (see Table 5). The CP Rank models typically achieve second and third place on these measures. The MIP models are next in terms of proofs of optimality/infeasibility but trail the CP Interval models in terms of primal gap. This poor MIP performance is due to the solver running out of memory for some instances with $n \geq 100$, while none of the other solvers exhibited memory issues. Finally, DIDP Del is the worst performing model in terms of proofs but outperforms the MIP models based on the quality of primal solutions.

Deletion vs. Addition Variants. As expected, the DIDP Add model is substantially better than DIDP Del model: using the deletion variant takes DIDP from the best performing exact approach to the worst. Consistent with Carmesin et al.'s DFS, Figure 13 shows that the ability to quickly find high quality solutions in DIDP Add is substantially impaired in the DIDP Del model. Similarly following expectations, the MIP and CP Rank models exhibit minor differences in performance between their deletion and addition variants in Figures 12 and 13. Interestingly, CP Interval Add performs substantially better than CP Interval Del, with the quick improvement in primal solution quality again likely a key factor. We speculate that in solving a scheduling model, CP Optimizer may be employing primal heuristics that build the solution chronologically.

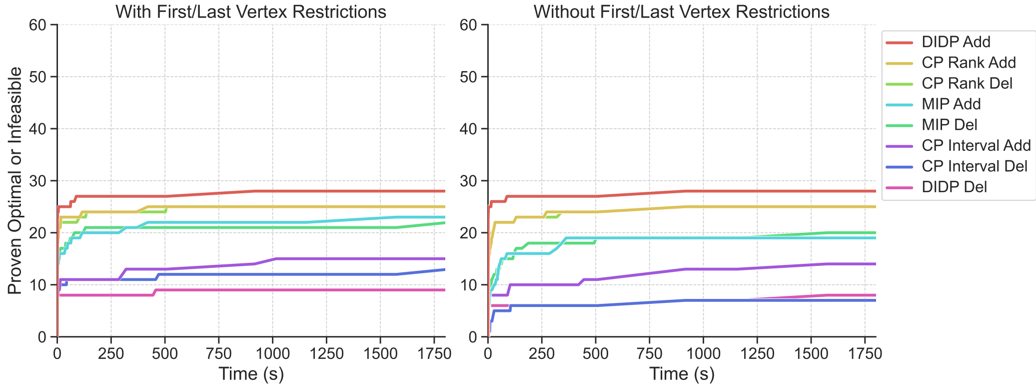

Figure 12 Instances proven optimal or infeasible over time for all exact models. Recall that 21 of the instances are infeasible and the rest admit feasible solutions.

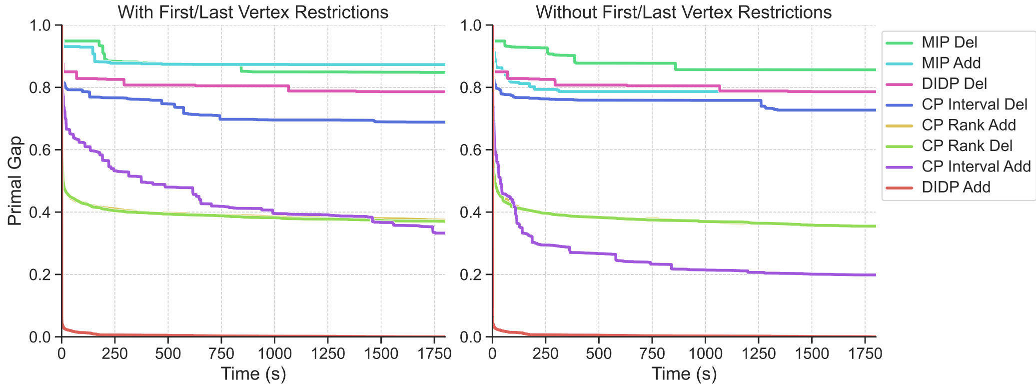

Figure 13 Mean primal gap to best known solution over time of all exact models, averaged over 39 feasible instances. Note the the two CP Rank plots coincide.

Table 2 provides more detailed data. For a given approach (e.g., CP Rank), we identified the problem instances for which both the deletion and addition variants proved optimality or infeasibility. We provide the shifted geometric mean (with a shift of 1) and the arithmetic mean of solver-specific measures of the search effort in Table 2. With the exception of MIP Add having almost four times the number of simplex iterations in geometric mean as MIP Del, CP Rank and MIP models show minor differences. The CP Interval models show a larger reduction in most measures of search effort when using the addition variant, consistent with the overall performance results. DIDP shows a substantial difference in the search effort (e.g., 4 orders of magnitude in the arithmetic means of the number of generated nodes), again consistent with the overall performance.

The Impact of First/Last Restrictions. Against our expectations, the inclusion of the first/last vertex restrictions appears to have either no impact or a small positive one on all models in terms of proving optimality or infeasibility. In terms of primal gap, however, both MIP Add and, to a greater extent, CP Interval Add improve when the redundant restriction constraints are *not* present. We speculate that the lack of restrictions allows some primal heuristics to perform better in these solvers. Further analysis is needed to understand the behavior of the models both with and without the first/last restrictions.

Table 2 The arithmetic and shifted geometric mean with shift of 1 of the solver specific search space measures over problems where each pair proved optimality or infeasibility for models without first/last vertex processing. The number in parentheses is the number of such instances.

		Geometric Mean		**Arithmetic Mean**	
Model	**Measure**	Deletion	Addition	Deletion	Addition
CP Rank (25)	No. Branches	111	139	333 006	239 412
	No. Fails	113	139	157 413	113 665
CP Interval (16)	No. Branches	1104	1069	1 288 442	795 770
	No. Fails	1038	1044	870 779	554 633
DIDP (9)	Nodes Generated	570	23.3	5 332 782	502
	Nodes Explored	518	21.9	4 630 268	421
MIP (21)	Nodes Explored	2.96	2.68	799	787
	Simplex Iterations	5.13	19.7	107 668	109 639

5.2 Experiment 2: Comparison with Previous Approaches

Problem Set. The second experiment uses the 11 instances with sizes from 14 to 1084 vertices adapted from TSPLIB [19] that Carmesin et al. [2] used for a detailed analysis.

Experimental Set-up. We use the same hardware and software as used in Experiment 1 for the single-threaded experiments. Unlike Experiment 1, following Carmesin et al., the time limit for each instance is $10n$ seconds, where n is the number of vertices. We also run each model using 8 threads with a memory limit of 32 GB to better compare with the settings of previous work. For Exact Init., following the literature, we run the single and multi-threaded versions 50 times for each instance and report the mean results.

The best performing approaches in the literature are the *Exact Init.* approach of Carmesin et al. [2] which employs a metaheuristic that is warm-started with the addition DFS approach and the *GRASP* of Woller et al. [21]. As the source code for Exact Init. is provided in the authors' repository,[2] we run it on our hardware based on the published hyper-parameter settings. We rely on the published results for the GRASP approach for which the source code is not available. The mean results for GRASP were back calculated given the gap between the mean results for the Exact Init. We compare the existing state of the art to the three best models in Experiment 1: CP Interval Add without first/last vertex processing, CP Rank Add with first/last vertex processing, and DIDP Add with first/last vertex processing.

Results. Table 3 presents the single-threaded results run on our hardware. DIDP Add dominates all other approaches by a considerable amount.

Moving to eight threads (Table 4), all approaches show some improvement compared to their single-thread results but the relative performance is the same, with DIDP Add again being dominant. In comparison to the literature, DIDP Add out-performs the reported results for Exact Init. on all problem instances and the reported results for GRASP up to and including size $n = 160$. For the three larger instances, GRASP finds the best solutions.

Note that the berlin52-10.4 instance was proved infeasible by the exact solvers. However, the published results for both Exact Init. and GRASP show that a feasible solution was found. We checked the solution published by Carmesin et al. and confirmed that it uses a

[2] https://imr.ciirc.cvut.cz/Research/TSPSD

Table 3 The primal bounds obtained by the exact approaches and mean primal bound for the heuristic algorithms, all run on a single thread for $10n$ seconds with a memory limit of 8 GB. 'w.' indicates the model was run with first/last vertex restrictions, while 'w.o.' indicates the model was run without them. The '-' symbol indicates the instances was proved infeasible, while 't.o.' is a time-out, and 'm.o.' indicates a memory-out.

Instance	CP Rank Add w.	CP Interval Add w.o.	DIDP Add w.	Exact Init.
burma14-3.1	**52**	**52**	**52**	**52**
ulysses22-5.5	**141**	**141**	**141**	**141**
berlin52-10.4	-	t.o.	-	t.o.
berlin52-13.2	22 810	17 045	**15 331**	19 741
eil101-27.5	2 270	1 382	**1 183**	1 806
gr202-67.3	1 485	880	**777**	1 402
lin318-99.3	238 643	t.o.	**94 776**	312 641
fl417-160.6	213 448	t.o.	**22 789**	264 009
d657-322.7	465 012	t.o.	**85 547**	414 544
rat783-481.4	89 614	m.o.	**13 458**	62 679
vm1084-848.9	5 517 296	m.o.	**366 429**	2 667 016

Table 4 The primal bounds obtained by the exact approaches and mean primal bound for the heuristic algorithms, all run with 8 threads for $10n$ seconds with a memory limit of 32 GB. The '*' symbol indicates an invalid solution. See Table 3 for the meaning of the remaining symbols.

Instance	CP Rank Add w.	CP Interval Add w.o.	DIDP Add w.	Exact Init.	Exact Init. [2]	GRASP [21]
burma14-3.1	**52**	**52**	**52**	**52**	**52**	**52**
ulysses22-5.5	**141**	**141**	**141**	**141**	166	**141**
berlin52-10.4	-	-	-	t.o.	25 741*	26 231*
berlin52-13.2	20 025	17 438	**15 265**	18 908	17 835	18 852
eil101-27.5	2 153	1 416	**1 187**	1 655	1 513	1 484
gr202-67.3	1 446	835	**777**	1 152	849	870
lin318-99.3	223 836	t.o.	**93 660**	225 803	110 888	105 787
fl417-160.6	119 525	t.o.	**22 272**	27 259	27 259	25 787
d657-322.7	322 985	t.o.	84 257	264 829	85 347	**82 531**
rat783-481.4	69 212	t.o.	13 763	36 148	13 833	**12 906**
vm1084-848.9	5 756 755	m.o.	358 163	913 184	326 067	**305 525**

deleted edge and thus is invalid. In our execution, the Exact Init. algorithm, correctly, did not find any feasible solutions for berlin52-10.4.

6 Conclusion

In this paper, we investigated three model-and-solve paradigms for the Traveling Salesperson with Self-Deleting graphs (TSP-SD), a problem introduced by Carmesin et al. [2]: two constraint programming (CP) models based, respectively, on ranking and scheduling of vertex visits, a domain-independent dynamic programming (DIDP) model, and a mixed integer programming (MIP) model. We experimented with four variants of each model, constraining them to find forward sequences with edge deletion or backward sequences with edge addition, and with or without redundant constraints that restricted the start and end vertices.

Our numerical results showed that DIDP solving the addition variant of the problem significantly outperformed all the other exact models, performed better than the state-of-the-art heuristic methods on smaller and medium instances, but trailed the best heuristic approach in terms of solution quality on the largest of the tested instances.

Reformulating the problem to add edges rather than delete them showed little impact on the rank-based CP model and the MIP model but had modest and large positive impact, respectively, for the scheduling-based CP model and the DIDP model.

Our primary direction for future work is to generalize the problem to allow edges to be added and deleted by vertex visits. We expect this problem to be more challenging and, given the poor performance of DIDP on the deletion variant, that CP models may prove superior to the other approaches. We also plan to investigate model-based approaches to more complex state-dependent problems such as scheduling with time- or sequence-dependent costs [1, 3].

References

1 Matan Atsmony, Baruch Mor, and Gur Mosheiov. Single machine scheduling with step-learning. *Journal of Scheduling*, 27(3):227–237, June 2024. doi:10.1007/s10951-022-00763-5.

2 S. Carmesin, D. Woller, D. Parker, M. Kulich, and M. Mansouri. The Hamiltonian Cycle and Travelling Salesperson problems with traversal-dependent edge deletion. *Journal of Computational Science*, 74:102156, 2023. doi:10.1016/j.jocs.2023.102156.

3 Erik Diessel and Heiner Ackermann. Domino sequencing: Scheduling with state-based sequence-dependent setup times. *Operations Research Letters*, 47(4):274–280, July 2019. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167637719300598, doi:10.1016/j.orl.2019.04.004.

4 Y. Dumas, J. Desrosiers, E. Gelinas, and M.M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2):367–371, 1995. URL: http://www.jstor.org/stable/171843.

5 T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989. URL: https://www.sciencedirect.com/science/article/pii/0167637789900023, doi:10.1016/0167-6377(89)90002-3.

6 F. Focacci, A. Lodi, and M. Milano. A hybrid exact algorithm for the tsptw. *INFORMS Journal on Computing*, 14(4):403, Fall 2002.

7 A.R. Güner, A. Murat, and R.B. Chinnam. Dynamic routing for milk-run tours with time windows in stochastic time-dependent networks. *Transportation Research Part E: Logistics and Transportation Review*, 97:251–267, 2017. URL: https://www.sciencedirect.com/science/article/pii/S1366554515301198, doi:10.1016/j.tre.2016.10.014.

8 Christoph Hansknecht, Imke Joormann, and Sebastian Stiller. Dynamic Shortest Paths Methods for the Time-Dependent TSP. *Algorithms*, 14(1):21, January 2021. URL: https://www.mdpi.com/1999-4893/14/1/21, doi:10.3390/a14010021.

9 Jahangirnagar University, E. Islam, M. Sultana, and F. Ahmed. A Tale of Revolution: Discovery and Development of TSP. *International Journal of Mathematics Trends and Technology*, 57(2):136–139, May 2018. doi:10.14445/22315373/IJMTT-V57P520.

10 W.-K. Ku and J. C. Beck. Revisiting off-the-shelf mixed integer programming and constraint programming models for job shop scheduling. *Computers & Operations Research*, 73:165–173, 2016.

11 M. Kulich, D. Woller, S. Carmesin, M. Mansouri, and L. Přeučil. Where to place a pile? In *2023 European Conference on Mobile Robots (ECMR)*, pages 1–7, 2023. doi:10.1109/ECMR59166.2023.10256330.

12 R. Kuroiwa. *Domain-Independent Dynamic Programming.* PhD thesis, University of Toronto, 2024.

13 R. Kuroiwa and J.C. Beck. Domain-independent dynamic programming: Generic state space search for combinatorial optimization. In *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS2023)*, pages 245–253, 2023.

14 Raphael Kühn, Christian Weiß, Heiner Ackermann, and Sandy Heydrich. Scheduling a single machine with multiple due dates per job. *Journal of Scheduling*, 27(6):565–585, December 2024. doi:10.1007/s10951-024-00825-w.

15 N. Lipovetzky, C. Burt, A. Pearce, and P. Stuckey. Planning for mining operations with time and resource constraints. *Proceedings of the International Conference on Automated Planning and Scheduling*, 24(1):404–412, May 2014. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13666, doi:10.1609/icaps.v24i1.13666.

16 Yiqing L. Luo and J. C. Beck. Packing by scheduling: Using constraint programming to solve a complex 2d cutting stock problem. In Pierre Schaus, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 249–265, Cham, 2022. Springer International Publishing.

17 J.J. Miranda-Bront, I. Méndez-Díaz, and P. Zabala. An integer programming approach for the time-dependent tsp. *Electronic Notes in Discrete Mathematics*, 36:351–358, 2010. ISCO 2010 - International Symposium on Combinatorial Optimization. URL: https://www.sciencedirect.com/science/article/pii/S1571065310000466, doi:10.1016/j.endm.2010.05.045.

18 J.-C. Picard and M. Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, 26(1):86–110, 1978. URL: http://www.jstor.org/stable/169893.

19 G. Reinelt. Tsplib—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991. arXiv:https://doi.org/10.1287/ijoc.3.4.376, doi:10.1287/ijoc.3.4.376.

20 D. Woller. URL: https://imr.ciirc.cvut.cz/Research/TSPSD.

21 D. Woller, M. Mansouri, and M. Kulich. Making a complete mess and getting away with it: Traveling salesperson problems with circle placement variants. *IEEE Robotics and Automation Letters*, 9(10):8555–8562, 2024. doi:10.1109/LRA.2024.3445817.

22 J. Zhang and J.C. Beck. Domain-independent dynamic programming and constraint programming approaches for assembly line balancing problems with setups. *INFORMS Journal on Computing*, 2024. in press, published onlinne October 2024. arXiv:https://doi.org/10.1287/ijoc.2024.0603, doi:10.1287/ijoc.2024.0603.

A Detailed Results

Table 5 Exact models run for 30 minutes with an 8 GB memory limit over 39 randomly generated feasible instances. The time to first solution (TTFS) in seconds, initial, and final primal gaps are averaged over all 39 instances. If the model experienced a memory-out prior to obtaining a feasible solution, the time was taken as 1800s, and the primal and optimal gap were taken as 100%.

	Model	Feasible	Best	Opt	TTFS	Primal Gap Initial	Primal Gap Final	Optimality Gap Final
Add	CP Rank	39	6	4	1.4	59.1%	35.0%	81.7%
	CP Interval	31	6	6	548	55.0%	31.2%	74.9%
	DIDP	39	39	9	<0.01	22.2%	0.0%	32.9%
	MIP	5	4	3	1 577	91.2%	87.4%	88.0%
Del	CP Rank	39	6	4	1.1	58.9%	34.7%	78.7%
	CP Interval	14	7	6	1 200	79.2%	69.0%	80.9%
	DIDP	9	5	2	1 421	80.8%	78.7%	84.4%
	MIP	6	4	2	1 559	88.8%	84.9%	85.8%
Without First/Last Vertex Restriction								
Add	CP Rank	39	4	4	5.9	59.4%	35.6%	83.3%
	CP Interval	36	5	6	238	47.4%	20.1%	78.8%
	DIDP	39	38	9	<0.01	22.9%	0.1%	33.0%
	MIP	11	4	4	1 395	86.0%	78.5%	83.9%
Del	CP Rank	39	6	4	5.3	58.9%	35.6%	82.6%
	CP Interval	10	3	4	1 372	82.7%	74.8%	88.3%
	DIDP	9	5	2	1 422	80.9%	78.7%	84.5%
	MIP	6	3	2	1 609	88.1%	85.7%	88.0%

Table 6 Performance of exact models, averaged over the 21 infeasible instances. Time taken as 1800s if the instance could not be proven infeasible.

	Model	With First/Last Vertex Restrictions Infeasible	With First/Last Vertex Restrictions Time (s)	Without First/Last Vertex Restrictions Infeasible	Without First/Last Vertex Restrictions Time (s)
Add	CP Rank	21	1.2	21	9.4
	CP Interval	11	929	8	1 127
	DIDP	21	<0.01	21	<0.01
	MIP	20	108	15	543
Del	CP Rank	21	2.9	21	10.7
	CP Interval	10	1 021	3	1 543
	DIDP	7	1 222	6	1 350
	MIP	19	179	18	321

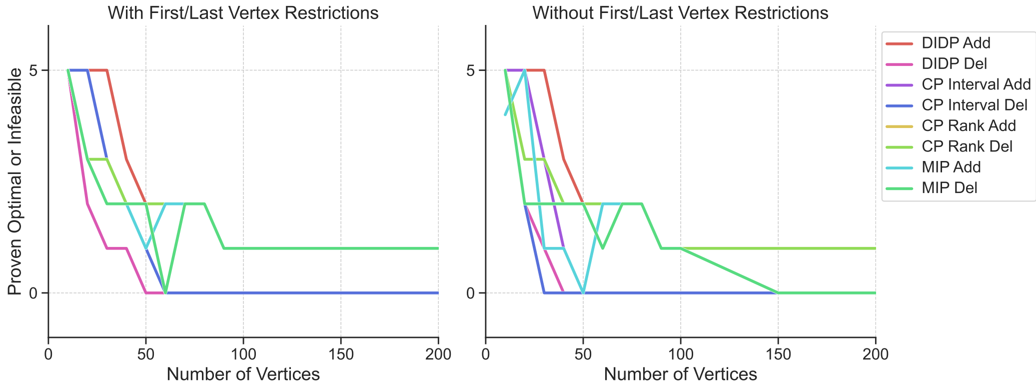

Figure 14 Instances proven optimal or infeasible for all exact models by the number of vertices in the instance for Experiment 1. Recall that 21 of the instances are infeasible and the rest admit feasible solutions, and that there are 5 instances per n.

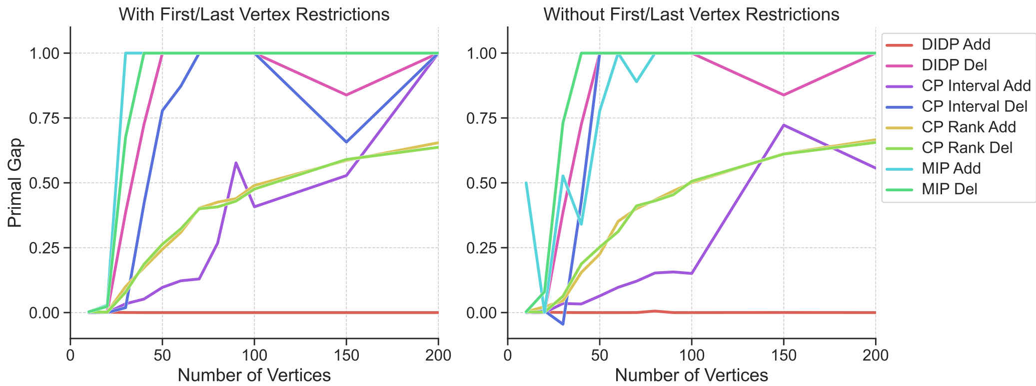

Figure 15 Mean gap to best known solution (i.e., primal gap) by the number of vertices in the instance of all exact models for Experiment 1, averaged over 39 feasible instances.

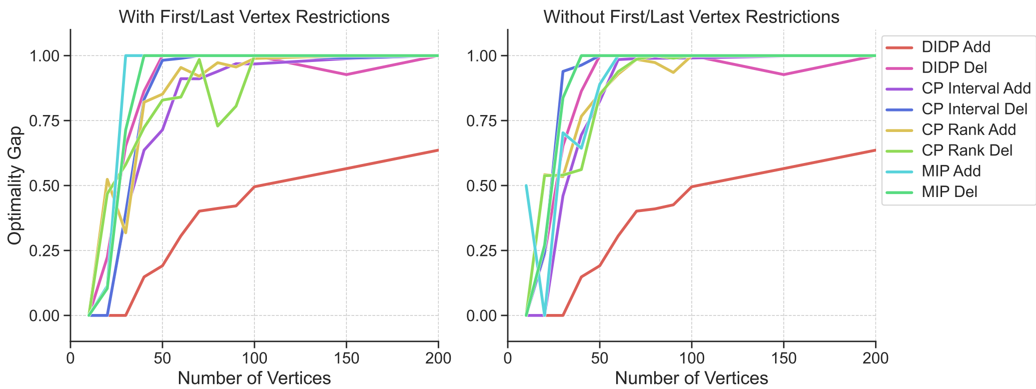

Figure 16 Mean optimality gap by the number of vertices in the instance of all exact models for Experiment 1, averaged over 39 feasible instances.