Decision Diagrams for Solving Traveling Salesman Problems with Pickup and Delivery in Real Time

Ryan J. O'Neil[a,b], Karla Hoffman[a]

[a] *Department of Systems Engineering and Operations Research, George Mason University, Fairfax, Virginia 22030*
[b] *Decision Engineering Department, Grubhub, Chicago, Illinois 60602*

Abstract

The Traveling Salesman Problem with Pickup and Delivery seeks a minimum cost path with pickups preceding deliveries. It is important in on-demand last-mile logistics, such as ride sharing and meal delivery. We examine the use of low-width Decision Diagrams in a branch-and-bound with and without Assignment Problem inference duals as a primal heuristic for finding good solutions within strict time budgets. We show these diagrams can be more effective than similarly structured hybrid Constraint Programming techniques for real-time decision making.

Keywords: traveling salesman problem pickup and delivery; decision diagrams; heuristics; real-time optimization

1. Introduction

The Traveling Salesman Problem with Pickup and Delivery (TSPPD) is the search for a minimum cost Hamiltonian circuit connecting a set of pairwise pickup and delivery locations where pickups must precede their associated deliveries [1, 2]. The TSPPD is the single-vehicle version of the Pickup and Delivery Problem (PDP), which includes such instances as the many-to-many Dial-A-Ride Problem (DARP) of Psaraftis (1980) [3] and Bertsimas et al. (2018) [4],

Email addresses: roneil1@gmu.edu, roneil@grubhub.com (Ryan J. O'Neil), khoffman@gmu.edu (Karla Hoffman)

and the Meal Delivery Routing Problem (MDRP) of Reyes et al. (2018) [5]. TSPPDs and their variants play an increasingly important role in industrial routing applications, as witnessed by a proliferation of ride hailing and sharing companies, as well as on-demand delivery service providers for everything from groceries, alcoholic beverages, and meals, to snacks and convenience store items.

Our interest is on the use of exact optimization methods for dynamic real-time logistics, where optimization procedures are time limited and "good" solutions must be provided quickly, frequently within seconds or milliseconds. Such problems tend to have practical limitations associated with routing as well. For example, delivery trucks are often limited to routes of fewer than 100 stops due to physical considerations, such as vehicle capacity [6]. Routes used for high volume restaurant delivery are even shorter due to the perishability of goods, and typically involve fewer than 10 stops.

In a prior study, we applied and compared different forms of hybrid optimization techniques to real-time TSPPD problems from observed meal delivery data at Grubhub [7]. We found that a circuit-based Constraint Programming (CP) formulation with a set-based precedence propagator, Sequential Closest Neighbor (SCN) branching, and an Assignment Problem (AP) inference dual is effective at finding good solutions (i.e. solutions within 10% of the true optimum) to TSPPD instances quickly, and that it works well as a warm start for Mixed Integer Programming (MIP) models. SCN branching builds a single tour starting at the origin node. It branches by adding the closest next feasible node to the end a partial tour on the left and not on the right [7].

This paper extends that work with techniques from Bergman et al. (2016) [8] and Cire and van Hoeve (2013) [9]. We use Decision Diagrams (DD) as a primal heuristic with and without an AP inference dual. We show that the breadth-first approximation employed by Decision Diagrams (DD) can quickly find very good solutions to TSPPDs, and that its performance often exceeds that of our CP-based formulation.

All of our DD configurations show median times to find solutions within 10% of optimality for our largest test instances (15 pairs) under 40 milliseconds,

while our CP implementation with an AP inference dual has a median time of 155 milliseconds. For all problems tested, the DD implementation provided better median times than those observed for our CP implementation. Our DD implementation is also able to find optimal solutions and prove optimality of solutions for more test instances than our CP model.

2. Background

The TSPPD is defined on an ordered set of pickup nodes $V_+ = \{+1, \ldots, +n\}$ and associated delivery nodes $V_- = \{-1, \ldots, -n\}$ such that $(+i, -i)$ form a request and $+i$ must precede $-i$ in a feasible route. V is defined as the union of V_+ and V_- with the addition of origin and destination nodes $\{+0, -0\}$. $E_\pm$ is the set of edges connecting $V_+ \cup V_-$. E is the union of $E_\pm$ with all feasible edges connecting to the origin and destination nodes. The graph $G = (V, E)$ includes all nodes and edges required to describe the TSPPD, i.e.,

$$V = \{+0, -0\} \cup V_+ \cup V_- \tag{1}$$

$$E = \{(+0, -0)\} \cup \{(+0, +i) \mid i \in V_+\} \cup \{(-0, -i) \mid i \in V_-\} \cup E_\pm \tag{2}$$

For modeling convenience, the edge $(+0, -0)$ must be included in any feasible solution. The set of edges E is defined such that it includes only feasible edges. That is, it is not possible that a TSPPD route begins with a delivery or ends with a pickup. Hybrid CP and Mixed Integer Programming (MIP) formulations of the TSPPD are provided in O'Neil and Hoffman (2018) [7].

DDs are represented as layered, acyclic, directed multigraphs connecting a *root* node r to a *terminal* node t. Each node in the diagram is a state with an associated cost, and is part of a layer. Layers correspond to the order in which decisions are made. Layers L_1 and L_{n+1} are singletons containing r and t, respectively. Each node in layer L_i connects to nodes in L_{i+1}, and solving an optimization problem corresponds to computing a shortest (or longest) path through the diagram from r to t. Bergman et al. (2016) [8] provide an extensive

treatment of optimization based on DDs. They define DDs using states and transition functions, based on a dynamic programming approach, i.e. based on Bellman's principle of optimality.

Formulation 1 defines a DD-based optimization model for the TSPPD, without explicitly referencing the r and t states. π is an ordered vector of $2(n+1)$ nodes representing a feasible TSPPD path through $2n$ pickup and delivery pairs from $+0$ to -0. The objective function (3) sums the cost of each arc traveled, represented as $c(\pi_i, \pi_{i+1})$. The first node in the path must be $+0$, as required by constraint (4). All subsequent nodes must be feasible given the current partial route from π_0 to π_i, requiring that π_{i+1} belong to the feasible domain of next nodes, D_i, as in constraint (5). Constraints (6, 7, 8) maintain feasibility of π by updating the domain of feasible next nodes, D_{i+1}. These require that the first node visited after $+0$ be a pickup, that pickups precede their associated deliveries, and that the last node visited be the end node -0.

$$\texttt{minimize} \quad \sum_{i=0}^{2n} c(\pi_i, \pi_{i+1}) \tag{3}$$

$$\texttt{subject to} \quad \pi_0 = +0 \tag{4}$$

$$\pi_{i+1} \in D_i \qquad \forall \quad i = 0, \ldots, 2n \tag{5}$$

$$D_0 = V_+ \tag{6}$$

$$D_{i+1} = \begin{cases} D_i \cup \{-\pi_i\} \setminus \{\pi_i\} & \text{if } \pi_i \in V_+ \\ D_i \setminus \{\pi_i\} & \text{if } \pi_i \in V_- \end{cases} \qquad \forall \quad i = 0, \ldots, 2n-2 \tag{7}$$

$$D_{2n} = \{-0\} \tag{8}$$

Formulation 1: Exact Decision Diagram

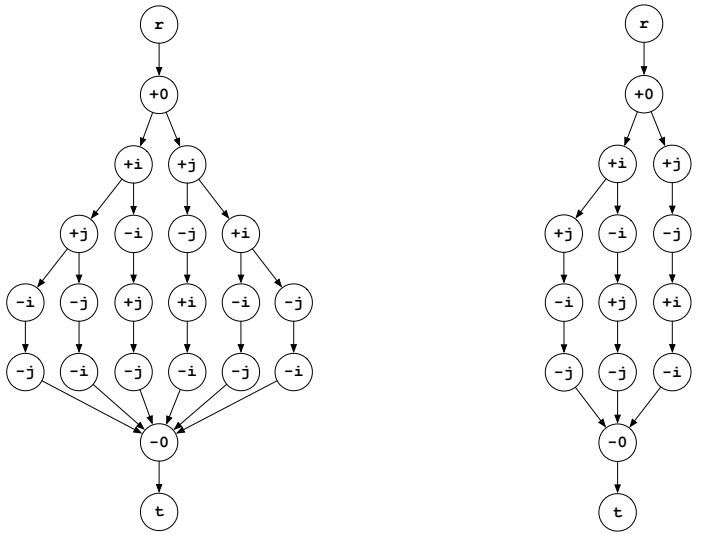
DDs can be used as general purpose optimizers and as primal heuristics without changes to their state formulations. Bergman et al. (2014) [10] adapt them to heuristics through the simple concept of a "restricted" diagram, which

selectively discards nodes in subsequent layers to maintain a maximum width. Bergman et al. (2016) [8] use restricted diagrams and "relaxed" diagrams (a form of relaxation dual) inside a branch-and-bound for optimization. We note that any dual can be substituted. We use an AP-based inference dual with reduced cost-based variable domain filtering in order to compare DDs directly with our previous work. Our previous testing showed that this dual worked well within the context of CP and we hope that it will work equally well within the DD framework.

It is simple to formulate a transition function for the TSPPD which is based on Cire et al. (2013) [9]. A sequential TSPPD state $s^j = (\pi_j, F_j)$ is a tuple containing the current partial path starting at $+0$ (π), and the domain of feasible next nodes (F). The domain of the root state is $\{+0\}$ and the domain at the node $+0$ is the set of pickups (V_+). Once a node is visited, it is removed from the F. If a pickup $+i$ is visited, its associated delivery $-i$ is added to F. Once all deliveries are visited, -0 must be visited, followed by t. As arcs are traversed, their costs are added to the current state cost.

The exact DD form of the TSPPD is restricted by defining a width and dropping the highest cost nodes from each layer prior to computing the next one. This keeps the diagram within a fixed amount of computing resources, while guaranteeing that feasible solutions are discovered. Feasibility is not guaranteed for restricted DDs in general, but is guaranteed for this particular model. Figure 1 shows possible exact and restricted DDs for a TSPPD with 2 pickup and delivery pairs, where the nodes represent the current location in the partial route.

Both the CP and DD models find feasible solutions very early in their search. The CP implementation uses a depth-first search to find a single initial solution, making greedy branching choices of the form $\texttt{next}_i = j \vee \text{next}_i \neq j$ at each node in the tree. In contrast, the DD implementation first constructs a restricted diagram layer by layer. It keeps more than one partial solution in each layer, according to the diagram width. This incurs some overhead depending on the width, and allows the DD to approximate a breadth-first search, which can



(a) Exact DD (b) Restricted DD

Figure 1: Example exact and restricted DDs for a two-pair TSPPD.

lead to discovery of better solutions early in the search. More details on our implementation are given in section 3.

3. Approach

We test two TSPPD solvers, one a CP model described in O'Neil and Hoffman (2018) [7], and the other being the sequential DD state transition described in section 2. The CP model uses a `circuit` constraint on a successor vector of decision variables called `next`. SCN branching first chooses a successor for $\texttt{next}_{+0}$ and then incrementally builds a single complete circuit from there. This is similar to how the DD sequential state transition works as they both maintain single partial feasible routes starting from +0. A set-based precedence propagator enforces that pickups precede deliveries. Optimization is accomplished with a branch-and-bound search, in which finding a new incumbent solution with cost z^* triggers the addition of a constraint $z < z^*$, where z is the objective value variable.

Our DD implementation also uses a branch-and-bound search, as described

in Bergman et al. (2016) [8]. We commence with an initial restricted diagram based at the root node r and solve a restricted DD from there. This provides a primal bound and incumbent solution. Instead of simply discarding states with high objective values during layer construction, we add them to a queue. This queue is sorted by descending layer depth and ascending dual then primal objective value, and feeds unexplored nodes to the branch-and-bound. Primal bounds from incumbent solutions enable filtering of states as diagrams are constructed and as nodes are removed from the queue.

In order to compare these models directly, we include an optional AP inference dual, as described in Focacci et al. (2002) [11]. This dual is solved at each node before the corresponding restricted DD. It also prunes nodes from the search tree using a dual bound and is used for reduced cost-based variable domain filtering. Our implementation is ported from Carpaneto et al. (1988) [12]. The AP dual has the advantage of executing quickly and updating incrementally in the search tree. Similarly, we focus our attention on low-width DDs which are fast at finding solutions and do not require much overhead.

4. Results

We test our CP and DD TSPPD models using symmetric instances constructed from pickup and delivery locations observed at Grubhub, along with expected travel times connecting location pairs. The test set has 10 instances per problem size. This allows us to gauge the performance of the models on realistic problems from meal delivery, in which pickup locations are likely to be clustered close together, and there may be many near optimal solutions that would have to be considered to prove optimality.

We evaluate the models according to the median and maximum times it takes them to find solutions within 10% of the true optimum, as determined by first solving the test instances to optimality. We use this as an indicator of a model's ability to find good solutions quickly, which is our primary concern in evaluating them for use in real-time logistics. It also suggests whether or not a

model is useful in providing a fast heuristic warm start to a MIP solver for full optimization, as in O'Neil and Hoffman (2018) [7].

Charts report the median and maximum of execution times for each problem size and model configuration with the vertical axis scaled to log time. Any model configuration that is not able to achieve a particular goal (e.g. finding a feasible solution, or finding a solution within 10% of optimality) for all instances of a given size is removed from consideration for that problem size.

Our CP implementation is written in C++14 and uses the Gecode 6.0.1 CP solver [13]. The DD model is written in Go 1.10. Our test set and source code for all model implementations are available for continued experimentation [14, 15, 16]. The test machine is a Lenovo X1 Carbon with a 4-core Intel Core i5 CPU and 16 GB of RAM. We test using a single thread, since such executions are deterministic and thus easier to interpret and compare.

Figure 2 shows the log of median and maximum times for the CP and DD solvers to find solutions within 10% of optimality for each of the various problem sizes. Vertical axes are scaled to the data. The DD model is tested with diagram widths from 1 to 20. Both models are tested with and without AP reduced cost-based variable domain filtering.

We observe that our DD implementation is always faster than the CP implementation with SCN branching at this metric, with the exception of instances with 11 and 14 pickup and delivery pairs, in which not all diagram widths were able to find solutions within 10% of optimality within 1000 seconds for each test instance. Very low-width diagrams appear to perform best, and there is often not much improvement provided by the AP inference dual.

We observe that our DD implementation outperforms the CP implementation with SCN branching in the majority of problem sizes and measures. AP inference duals start to have an important impact for finding solutions within 5% of optimality, finding optimal solutions, and proving optimality. Without these duals, we are not able to find solutions within 5% of optimality within 1000 seconds for all test instances larger than 10 pairs, find optimal solutions within 1000 seconds for all test instances larger than 8 pairs, or prove optimal-

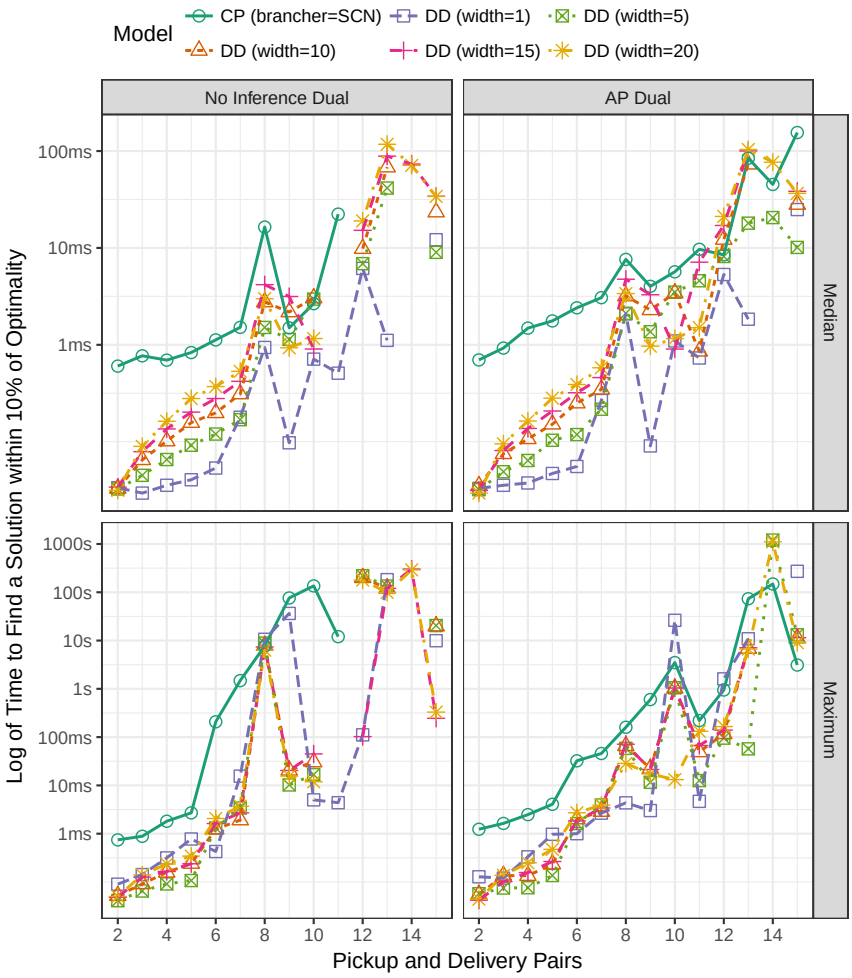


Figure 2: Median time in milliseconds for CP and DD models to find a solution within 10% of optimality for 10 problems of each size and worst cast (maximum) time in milliseconds to find a solution within 10% of optimality.

ity within 1000 seconds for all test instances larger than 8 pairs. AP inference duals speed up the finding of optimality solutions for problems with 8 pairs by DDs by roughly two orders of magnitude, and that impact is larger for proving optimality of instances with 7 and 8 pairs.

We expect the remarkable efficiency of the width-1 DD on many of the test instances is due to our state structure and the ability of low-width DDs to effectively use the priority queue. Our DD state implementation for the TSPPD requires little more than a pointer to a parent state and a feasible vector, and the priority queue is sorted by descending layer depth and ascending objective value [16]. For larger widths DDs, merging nodes provides obvious benefits, while for these very low-width diagrams the merging may be more costly than the basic search procedure.

5. Conclusions and Future Work

In the paper we present a DD approach to the solving TSPPDs in real-time logistics systems and compare it to an existing CP approach. For the class of problems that we are considering, time budgets for route optimization are on the order of milliseconds or, at most, seconds. We test these models against a variety of problem instance sizes, and use a range of widths in our DD implementation.

We find that our DD implementation is nearly always faster in median times than our CP baseline model at finding good solutions, as well as at finding optimal solutions and proving optimality. We also find that AP duals provide little benefit, but also cost little additional computing time, until problems become large enough for them to matter. At that point they speed up the solution time by orders of magnitude and extend the DD's ability to solve and optimize larger problems. Further, we find that increasing diagram width does not help much in median time to find good solutions, and can add computational overhead to the model. However diagrams with width of 5 or 10 can have better maximum times to find good solutions or optimal solutions, and tend to be able to prove optimality for larger instances faster. In summary, we recommend

low-width diagrams for this use case.

We believe these results are interesting and useful in that they measure the effectiveness of DDs against a model already known to be quite effective for providing real-time solutions and warm starting MIP models, and show that a simple DD frequently has better performance. We also show that the impact of an AP inference dual grows by orders of magnitude as problem size increase. Further work includes investigating how well DD implementations work as a warm start for a MIP algorithm, the incorporation of relaxed DDs enhanced with Lagrangian bounds in conjunction with the AP inference dual for pruning nodes in the search tree, and the use of additive bounding combining multiple duals to strengthen these bounds and inference procedures [17, 18].

References

[1] K. Ruland, E. Rodin, The Pickup and Delivery Problem: Faces and Branch-and-Cut Algorithm, Computers & Mathematics with Applications 33 (12) (1997) 1–13.

[2] I. Dumitrescu, S. Ropke, J.-F. Cordeau, G. Laporte, The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm, Mathematical Programming 121 (2) (2010) 269.

[3] H. N. Psaraftis, A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem, Transportation Science 14 (2) (1980) 130–154.

[4] D. Bertsimas, P. Jaillet, S. Martin, Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications, accepted for publication in Operations Research (2018).

[5] D. Reyes, A. Erera, M. Savelsbergh, S. Sahasrabudhe, R. J. O'Neil, The Meal Delivery Routing Problem, `http://www.optimization-online.org/DB_HTML/2018/04/6571.html` (2018).

[6] Y. Caseau, F. Laburthe, Solving Small TSPs with Constraints, in: ICLP, Vol. 97, 1997, p. 104.

[7] R. J. O'Neil, K. Hoffman, Exact Methods for Solving Traveling Salesman Problems with Pickup and Delivery in Real Time, http://www.optimization-online.org/DB_HTML/2017/12/6370.html (2018).

[8] D. Bergman, A. A. Cire, W.-J. van Hoeve, J. Hooker, Decision Diagrams for Optimization, Springer, 2016.

[9] A. A. Cire, W.-J. van Hoeve, Multivalued Decision Diagrams for Sequencing Problems, Operations Research 61 (6) (2013) 1411–1428.

[10] D. Bergman, A. A. Cire, W.-J. van Hoeve, T. Yunes, BDD-based heuristics for binary optimization, Journal of Heuristics 20 (2) (2014) 211–234.

[11] F. Focacci, A. Lodi, M. Milano, A Hybrid Exact Algorithm for the TSPTW, INFORMS Journal on Computing 14 (4) (2002) 403–417.

[12] G. Carpaneto, S. Martello, P. Toth, Algorithms and codes for the assignment problem, Annals of Operations Research 13 (1) (1988) 191–223.

[13] Gecode Team, Gecode: Generic constraint development environment (2006).
URL http://www.gecode.org

[14] Grubhub, TSPPD Test Instance Library, https://github.com/grubhub/tsppdlib (2018).

[15] R. J. O'Neil, TSPPD Hybrid Optimization Code, https://github.com/ryanjoneil/tsppd-hybrid (2018).

[16] R. J. O'Neil, TSPPD Decision Diagram Code, https://github.com/ryanjoneil/tsppd-dd (2018).

[17] J. Kinable, A. A. Cire, W.-J. van Hoeve, Hybrid Optimization Methods for Time-Dependent Sequencing Problems, European Journal of Operational Research 259 (3) (2017) 887–897.

[18] D. Bergman, A. A. Cire, W.-J. van Hoeve, Lagrangian Bounds from Decision Diagrams, Constraints 20 (3) (2015) 346–361.