Scalability and Performance of Decentralized Planning in Flexible Transport Networks

Van Heeswijk, Wouter
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
wouter.van.heeswijk@cwi.nl

La Poutré, Han
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
han.la.poutre@cwi.nl

Abstract

This paper addresses the planning of freight dispatch in flexible transport networks featuring multiple carriers. To deal with the computational challenges of the planning problem, we develop an Approximate Dynamic Programming (ADP) algorithm that utilizes neural network techniques to learn dispatch policies. We test whether dispatch policies learned autonomously by carrier agents (based on local information) match the quality of policies learned by a central planner (based on full network information). Numerical experiments show that the policies yield solutions of comparable quality for small instances, yet the decentralized approach is capable to scale to larger instances. Finally, the ADP policies are compared to four benchmark policies, which are all significantly outperformed.

1 Introduction

In recent years, freight transport has become increasingly difficult to organize due to higher service standards, i.e., smaller order volumes, more frequent shipments, and faster deliveries. To maintain a high transport efficiency, individual carriers (transport companies) often must collaborate and utilize physical decoupling points (e.g., transfer hubs) to form flexible multi-segment transport networks that facilitate the bundling and unbundling of fractional loads. Such networks enable consolidation of transport jobs with geographically dispersed origins and/or destinations. The transport flows are typically controlled by a central planner. The problem of selecting the next route segment and dispatch time for individual transport jobs is known as the dispatching problem.

A downside of centralized planning is the large amount of information that needs to be shared by carriers. Combined with the stochastic and dynamic nature of the planning problem, it is challenging to find good solutions within limited time. Also, the objectives of the individual (possibly competing) carriers may diverge from those of the central planner. In contrast, decentralized planning enables carriers to pursue their own goals and eliminates the need to share information.

This paper addresses the dispatching problem in a fundamental network setting. Our main contributions are as follows. We learn dispatch policies in both centralized and decen-

tralized settings and compare the performance of these policies, taking into account operational efficiency, timely delivery, and distance covered. Numerical experiments demonstrate that decentralized planning achieves solutions of a quality similar to centralized planning, while exploiting the vast computational benefits of decentralized planning. For this purpose we develop an Approximate Dynamic Programming (ADP) algorithm, using neural network techniques to learn dispatch policies based on observed values. Of particular importance is the scalability of our decentralized solution to larger instances, as in centralized planning the number of possible actions increases exponentially with the number of agents.

2 Related literature

This section concisely presents literature on various topics related to the problem studied in this paper. In contrast to our work, most studies consider centralized planning.

First, we address the topic of flexible multi-segment transport planning. This entails responding to new transport jobs that are revealed dynamically over time and adjusting the planning to utilize arising consolidation opportunities. The computational challenge lies in the vast amount of possible solutions that typically emerge in combinatorial optimization problems. Common solution approaches are local search algorithms (e.g., Ferucci et al. [1]) and constructive algorithms (e.g., Van Heeswijk et al. [2]).

The Delivery Dispatching Problem and the Service Network Design problem are problem classes concerned with the selection and dispatch timing of transport services. In stochastic and dynamic settings, scenario sampling or value function approximation (which true value depends on the underlying stochastic process) are the most common solution methods [3]. Examples of both approaches may be found in Lium et al. [4] and Van Heeswijk et al. [5], respectively.

As mentioned in Section 1, large transport networks are difficult to control centrally due to the required degree of information sharing and potential misalignment of objectives. Research on decentralized transport planning is limited. Mes et al. [6] heuristically compare centralized and decentralized planning, and argue that decentralized planning may perform at least as well as their centralized benchmark policy. Jung et al. [7] reach similar conclusions, stating that their decentralized planning method approximates the performance of an idealized centralized planning.

The final topic that we address is learning. Transport problems with stochastic and dynamic problems can generally be modeled as Markov decision models, but these are typically too large to solve exactly [8]. Instead of computing the optimal decision policy, we often settle for finding near-optimal policies. Stochastic modeling frameworks such as Approximate Dynamic Programming (ADP) are often used for this purpose, generally considering a single planner in transport settings. Based on sampled observations, the objective is to estimate the true value function of the problem, enabling to quantify the downstream effects of actions [9].

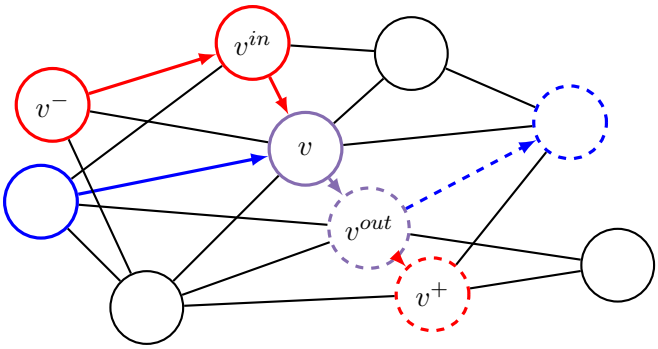

Figure 1: Example of network. The red vertex pair (v^-, v^+) represents the origin and destination for one job, the blue vertex pair for another. Following the colored paths, the jobs might be consolidated on the purple arc connecting v to v^{out}.

3 Model formulation

This section presents the formal definition of the Markov decision problem that we aim to solve. We describe the network, problem state, action definition, transition function, and value function from the perspective of a single carrier agent that operates from a predefined vertex. The model is solved over an infinite planning horizon $\mathcal{T}$ that consists of decision epochs separated by equidistant time intervals.

3.1 Network

Let $\{\mathcal{V}, \mathcal{A}\}$ be an undirected and strongly connected graph. Figure 1 provides an illustrative network example with its most essential notations. The role of the vertices $v \in \mathcal{V}$ in the model is twofold. Every vertex represents a potential origin or destination of a transport job; a vertex pair (v^-, v^+) signifies the locations where the job is generated and where it should be delivered. In addition, each vertex also represents a carrier agent that may dispatch jobs to directly connected vertices.

The arcs $a \in \mathcal{A}$ represent transport services between two distinct vertices. Transport units (e.g., containers) filled with one or more jobs are moved via the arcs. If the cumulative job volume exceeds any positive integer number, multiple units are simultaneously dispatched. The set $\mathcal{V}_v \subset \mathcal{V}$ represents the set of vertices that may be reached directly from v and vice versa. We use the notation $v^{in} \in \mathcal{V}_v$ when referring to jobs transported from a vertex v^{in} to v and $v^{out} \in \mathcal{V}_v$ for jobs transported from v to a vertex v^{out}.

The following simplifications are made. First, the time intervals between adjacent decision epochs are assumed to be sufficiently large to transport jobs between any two connected vertices. Second, there are no handling times or capacity constraints at the vertices. Third, we define homogeneous transport services, transport units with a capacity of 1, and an unlimited number of transport units at each vertex.

3.2 State description

Transport jobs are randomly generated at some vertex $v^- \in \mathcal{V}$ and must be transported to some vertex $v^+ \in \mathcal{V} \setminus \{v^-\}$. Each job has a volume $l \in \mathcal{L}$, with $\mathcal{L} = \{\frac{1}{k}, \ldots, \frac{k}{k}\}$ and $k \in \mathbb{N}_{\geq 1}$; a volume of 1 equals the capacity of a transport unit. The due time at or before which the job should be delivered at v^+ – expressed relative to the current decision epoch (i.e., decreasing over time) – is denoted by t^+. Let $\mathcal{T}^+ = [0, t^{+,max}] \cap \mathbb{N}$ be the set of times from which t^+ may be drawn. The due time is a soft constraint, early dispatch is rewarded however. The variable t^{hold} is used to compute rewards for timely dispatch and indicates how long the job has been held at the current vertex v. The maximum allowed holding time is $t^{hold,max}$, yielding a set of holding times $\mathcal{T}^{hold} = [0, t^{hold,max}] \cap \mathbb{N}$. If a job with $t^{hold} = t^{hold,max}$ is not dispatched, it is considered a failed job and removed from the system; the agent incurs a penalty for this. Whenever a job arrives at a new vertex, we reset t^{hold} to 0.

We define a job type as a unique combination of the following properties: the due time t^+, the holding time t^{hold}, the job volume l, the current vertex v, and the destination vertex v^+. Let $(t^+, t^{hold}, l, v, v^+)$ be a job type and let $I_{t^+,t^{hold},l,v,v^+} \in \mathbb{N}$ be the number of jobs of that type in the system. The state of the system for agent v is defined by

$$I_v = (I_{t^+,t^{hold},l,v,v^+})_{\forall (t^+,t^{hold},l,v^+) \in \mathcal{T}^+ \times \mathcal{T}^{hold} \times \mathcal{L} \times \mathcal{V}} \quad . \tag{1}$$

The set of all possible states for agent v is denoted by $\mathcal{I}_v$.

3.3 Action description

The actions that may be performed by carrier agent v depend on I_v; we denote a single action by $x_v(I_v)$ and the set of feasible actions by $\mathcal{X}_v(I_v)$. Essentially, the carrier agent must decide how many jobs per type to dispatch and to which vertex $v^{out} \in \mathcal{V}_v$ they are dispatched. Each decision variable $x_{t^+,t^{hold},l,v,v^{out},v^+}$ represents a nonnegative integer number of jobs of a given type to be transported via a given arc. Compared to the state variable, the subscript v^{out} is added to the decision variable, i.e., the outbound vertex to which the job is assigned. Jobs are not allowed to be shipped to vertices if it increases the distance to their destination. Furthermore, each job may be assigned to at most one outbound arc. The action for carrier agent v given state I_v is defined as

$$x_v(I_v) = (x_{t^+,t^{hold},l,v,v^{out},v^+})_{\substack{\forall (t^+,t^{hold},l,v^{out},v^+) \in \\ \mathcal{T}^+ \times \mathcal{T}^{hold} \times \mathcal{L} \times \mathcal{V}_v \times \mathcal{V}}} \quad . \tag{2}$$

3.4 Transition function

This section describes the transition function that determines the state change when moving from one decision epoch to the next. For carrier agent v, the transition depends on the action x_v, the actions $x_{v^{in}}, \forall v^{in} \in \mathcal{V}_v$ (which are not directly observable by agent v), and the set of new jobs with origin $v^- = v$ generated according to a stochastic process. We define the latter as follows. Let $\tilde{I}_{t^+,0,l,v,v^+} \in \mathbb{N}$ be a random variable describing the number of new jobs

of type $(t^+, 0, l, v, v^+)$ generated during the interval separating the two adjacent decision epochs. The random variable vector $W_v = (\tilde{I}_{t^+,0,l,v,v^+})_{\forall(t^+,l,v^+)\in\mathcal{T}^+\times\mathcal{L}\times\mathcal{V}}$ denotes the jobs generated at v during the time interval. We use ω_v to describe a potential combination of generated jobs and Ω_v as the set of all possible combinations of generated jobs. When W_v returns an outcome $\omega_v \in \Omega_v$, we define this realization by $W_v(\omega_v)$. From the perspective of agent v, the dispatch actions of adjacent agents are also a stochastic arrival process, as their states and actions cannot be observed. This is a crucial difference with central planning, in which the planner observes the states and actions of the complete network. To reflect the uncertainty in arrivals, we use the random variable $\tilde{x}_{t^+,0,l,v,v^+} \in \mathbb{N}$ to denote the number of job arrivals of a specific type stemming from vertices adjacent to v (i.e., a local observation at v), $\xi_v = (\tilde{x}_{t^+,0,l,v,v^+})_{\forall(t^+,l,v^+)\in\mathcal{T}^+\times\mathcal{L}\times\mathcal{V}}$ as a potential combination of job arrivals, Ξ_v as the set of all possible combinations of arrivals, X_v as the random variable vector of arrivals, and $X_v(\xi_v)$ as a realization.

We split the transition function into two parts. The first part is deterministic and describes the transition from I_v to the so-called post-decision state I_v^{post} [8]. This transition depends solely on the action taken by carrier agent v, describing the state before the newly generated jobs and before arrivals from adjacent vertices. The second part is stochastic and describes the transition from I_v^{post} to a new pre-decision state I'_v and requires the stochastic realizations of job generations $W_v(\omega_v)$ and job arrivals $X_v(\xi_v)$. We describe the first part of the transition function – defined by S^{post} – as follows:

$$I_v^{post} = S^{post}(I_v, x_v) \ , \tag{3}$$

s.t.

$$I^{post}_{t^+,t^{hold},l,v,v^{out},v^+} = I_{t^+,t^{hold},l,v,v^+} - \sum_{v^{out}\in\mathcal{V}_v} x_{t^+,t^{hold},l,v,v^{out},v^+} \tag{4}$$
$$\forall(t^+, t^{hold}, l, v^+) \in \mathcal{T}^+ \times \mathcal{T}^{hold} \times \mathcal{L} \times \mathcal{V} \ .$$

We proceed with the second part of the transition function, which we denote by S^{step}. This part processes the arrivals stemming from the decisions made by the adjacent carrier agents $v^{in} \in \mathcal{V}_v$ – reflected by X_v – and the newly generated jobs W_v. It also adjusts the indices t^+ and t^{hold} to reflect the time step to the next decision epoch. Note that t^+ cannot get smaller than 0, but that t^{hold} is always incremented when holding a job.

$$I'_v = S^{step}(I_v^{post}, W_v(\omega_v), X_v(\xi_v)) \ , \tag{5}$$

s.t.

$$I_{t^+,t^{hold},l,v,v^+} = \tag{6}$$
$$\begin{cases} I^{post}_{t^+ +1,t^{hold}-1,l,v,v^+} + \sum_{v^{in}\in\mathcal{V}_v} \tilde{x}_{t^+,t^{hold},l,v^{in},v,v^+} + \\ \tilde{I}_{t^+,t^{hold},l,v,v^+} \quad \text{if } t^+ > 0 \ , \\ I^{post}_{t^+,t^{hold}-1,l,v,v^+} + \sum_{v^{in}\in\mathcal{V}_v} \tilde{x}_{t^+,t^{hold},l,v^{in},v,v^+} + \\ \tilde{I}_{t^+,t^{hold},l,v,v^+} \quad \text{if } t^+ = 0, t^{hold} < t^{hold,max} \ , \end{cases}$$
$$\forall (t^{hold}, t^+, l, v^+) \in \mathcal{T}^+ \times \mathcal{T}^{hold} \times \mathcal{L} \times \mathcal{V} \ .$$

3.5 Value function

Decentralized planning grants carrier agents autonomy with respect to the timing of dispatch and the selection of outbound arcs. The rewards should therefore be structured such that the functionality of the transport network is preserved, implying timely deliveries and efficient transport movements. The value function embeds three components corresponding to these goals, namely the transport costs $C(x_v) \in \mathbb{R}^+$ (a fixed distance-dependent number per transport unit dispatched), a distance reward $R^{dist}(x_v) \in \mathbb{R}^+$ (a number proportional to distance reduction towards destination), and a time reward $R^{time}(x_v) \in \mathbb{R}$ (a reward for quick dispatch, a penalty for holding jobs, and a penalty for failed jobs). We assign weights $w^{dist}, w^{time} \in \mathbb{R}^+$ to the two reward components.

The chosen action affects the rewards and costs incurred at subsequent decision epochs. Based on the knowledge of the stochastic process, the carrier agent can make an informed decision that considers the downstream effects of current actions. We use a factor $\gamma \in (0, 1)$ to discount future rewards.

A finite number of outcome states $I'_v \in \mathcal{I}_v$ may be reached from I_v, given action x_v and realizations $W_v(\omega_v)$ and $X_v(\xi_v)$. By multiplying the probability of realization $(W_v(\omega_v), X_v(\xi_v))$ with the discounted value of being in state I'_v and summing over all $(\omega_v, \xi_v) \in \Omega_v \times \Xi_v$, we may compute the expected downstream value for any state-action pair. The Bellman equation corresponding to the Markov decision model is defined by

$$V_v(I_v) = \max_{x_v \in \mathcal{X}_v(I_v)} \Bigg(w^{dist} \cdot R^{dist}(x_v) + w^{time} \cdot R^{time}(x_v) \tag{7}$$
$$-C(x_v) + \gamma \sum_{(\omega_v,\xi_v)\in\Omega_v\times\Xi_v} \mathbb{P}\left(W_v(\omega_v), X_v(\xi_v)\right) V_v(I'_v | I_v, x_v, \omega_v, \xi_v) \Bigg) \ ,$$
$$\forall I_v \in \mathcal{I}_v \ .$$

The solution of the Markov decision model is a policy $\pi_v : \mathcal{I}_v \mapsto \mathcal{X}_v$; the optimal policy for agent v may be found by recursively solving the Bellman equation for all states.

4 Solution method

The presented Markov decision model might theoretically be solved to optimality, but is computationally intractable for even moderate-sized problem instances. Therefore, we present an ADP-based solution method to solve the problem, using neural network techniques to estimate the true value function.

4.1 Algorithmic strategy

Solving Equation (7) poses major computational problems. The number of states, actions, and possible outcome states quickly becomes extremely large due to the many potential combinations of job arrivals and -generations. In our solution method, we address this problem as follows. Instead of computing the true value function $V_v(I_v)$ for each state, we compute an estimate $\tilde{V}_v(I_v)$. To learn this estimate, we perform a finite number of iterations. To avoid the need to evaluate all possible outcome states for a given state-action pair, we transform the stochastic problem in Equation (7) to a computationally more tractable deterministic problem. We replace the probabilistic term in the Bellman equation with a statistical expectation $\bar{V}_v(I_v^{post})$ that is based on sampled observations. After learning the dispatch policies, the deterministic maximization problem looks as follows:

$$\tilde{V}_v(I_v) = \max_{x_v \in \mathcal{X}_v(I_v)} \Bigg(w^{dist} \cdot R^{dist}(x_v) + w^{time} \cdot R^{time}(x_v) \\ -C(x_v) + \gamma \bar{V}_v(I_v^{post} | I_v, x_v) \Bigg) \ . \tag{8}$$

The statistical expectation $\bar{V}_v(I_v^{post})$ is obtained as follows. We select a state-action pair (I_v, x_v) to compute a post-decision state I_v^{post}, sample a realization $(W_v(\omega_v), X_v(\xi_v))$ to reach a new state I_v', and use the value that we observe at the new state (i.e., the downstream costs corresponding to the state in the preceding iteration) to update the estimated value corresponding to I_v^{post}. A drawback of this approach is that we must observe every state multiple times to converge to the true values, which is computationally prohibitive. Instead, we compute the expected value based on certain features of the post-decision state (e.g., aggregate volumes per destination) rather than the complete state description, allowing to estimate values for states that have not yet been observed. We further address this in Section 4.2. For the algorithmic outline, it suffices to introduce a generic function U that updates after each iteration from estimate $\bar{V}_{v,n-1}(I_v^{post})$ to $\bar{V}_{v,n}(I_v^{post})$, with $n \in [0, N] \cap \mathbb{N}$ being the iteration counter and each iteration corresponding to a discrete time step. The update is based on an observed value $\hat{v}_{v,n}$ that follows from solving Equation 8, replacing $\bar{V}_v(I_v^{post})$ with $\bar{V}_{v,n-1}(I_v^{post})$.

Solving Equation 8 always returns the best action given the prevailing estimates. Particularly in the early iterations, the estimates of the downstream values may be poor. To prevent getting stuck in a local optimum early on, we apply an epsilon-greedy exploration strategy that selects a random action from $\mathcal{X}_v(I_v)$ with probability $\epsilon \in [0, 1]$ and the best

known action with probability $1-\epsilon$. This strategy enables both to frequently observe state-action pairs with high rewards to better learn their value and to explore unobserved parts of the state space.

Algorithm 1 outlines the algorithmic strategy to obtain value estimates $\bar{V}_{v,N}(I_v^{post}), \forall I_v^{post} \in \mathcal{I}_v$. In each iteration $n \in [0, N]$, we apply the value-maximizing action $x_{v,n}$ (given current estimates) on state $I_{v,n}$ to obtain post-decision state $I_{v,n}^{post}$. To reach a new state $I_{v,n+1}$, we randomly sample a realization of jobs $W_v(\omega_v)$. The realization of job arrivals $X_v(\xi_v)$ – although stochastic from the perspective of the carrier agent – is computed based on the actions of the adjacent agents, i.e., $\xi_v = \sum_{v^{in} \in \mathcal{V}_v} x_{v^{in},n}$. Subsequently, applying $x_{v,n+1}$ on $I_{v,n+1}$ yields a new observation $\hat{v}_{v,n+1}$, we use this observation to update our estimate $\bar{V}_{v,n}(I_v^{post})$.

Algorithm 1 *ADP algorithm to estimate post-decision values and obtain the dispatch policy (based on Powell [8]). The symbol $\overset{R}{\longleftarrow}$ implies a random draw from a set.*

1: **initialize** $\bar{V}_{v,0}(I_v^{post}), \forall I_v^{post} \in \mathcal{I}_v$
2: $\epsilon \longleftarrow [0, 1]$
3: $n \longleftarrow 1$
4: $I_{v,1} \overset{R}{\longleftarrow} \mathcal{I}_v$
5: **while** $n \leq N$ **do**
6: $\quad \epsilon_n \overset{R}{\longleftarrow} [0, 1]$
7: $\quad$ **if** $\epsilon_n > \epsilon$ **then**
8: $\quad\quad x_{v,n} \longleftarrow \underset{x_{v,n} \in \mathcal{X}_v(I_{v,n})}{\arg\max} \Big(w^{dist} \cdot R^{dist}(x_{v,n}) + w^{time} \cdot R^{time}(x_{v,n}) - C(x_{v,n}) + \gamma \bar{V}_{v,n-1}(I_v^{post} | I_{v,n}, x_{v,n}) \Big)$
9: $\quad$ **else if** $\epsilon_n \leq \epsilon$ **then**
10: $\quad\quad x_{v,n} \overset{R}{\longleftarrow} \mathcal{X}_v(I_{v,n})$
11: $\quad$ **end if**
12: $\quad I_{v,n}^{post} \longleftarrow S^{post}(I_{v,n}, x_{v,n})$
13: $\quad \hat{v}_{v,n} \longleftarrow w^{dist} \cdot R^{dist}(x_{v,n}) + w^{time} \cdot R^{time}(x_{v,n}) - C(x_{v,n}) + \gamma \bar{V}_{v,n-1}(I_{v,n}^{post})$
14: $\quad \bar{V}_{v,n}(I_v^{post}) \longleftarrow U\big(\bar{V}_{v,n-1}(I_v^{post}), I_{v,n-1}^{post}, \hat{v}_{v,n}\big)$
15: $\quad W_v(\omega_v) \overset{R}{\longleftarrow} \Omega_v$
16: $\quad X_v(\xi_v) \longleftarrow \sum_{v^{in} \in \mathcal{V}_v} x_{v^{in},n}$
17: $\quad I_{v,n+1} \longleftarrow S^{step}(I_{v,n}^{post}, W_v(\omega_v), X_v(\xi_v))$
18: $\quad n \longleftarrow n + 1$
19: **end while**
20: **return** $\bar{V}_{v,N}(I_v^{post}), \forall I_v^{post} \in \mathcal{I}_v$

4.2 Update function

Step 14 of Algorithm 1 performs the update $U\big(\bar{V}_{v,n-1}(I_v^{post}), I_{v,n-1}^{post}, \hat{v}_{v,n}\big)$, with U being an artificial neural network (ANN) that learns $\bar{V}_{v,N}(I_v^{post}), \forall I_v^{post} \in \mathcal{I}_v$. The ANN is composed as follows. It has an input layer based on $I_{v,n-1}^{post}$. The inputs are multiplied with

weights to provide input for a layer of hidden nodes that transforms the input. Subsequently, the hidden layer output is multiplied with weights to compute the expected value. Based on the difference between expected value $\bar{V}_{v,n-1}(I^{post}_{v,n-1})$ and observation $\hat{v}_{v,n}$, the ANN weights are updated [10].

We discuss the ANN update procedure in more detail. Suppose that at iteration $n-1$, we select a certain action that yields post-decision state $I^{post}_{v,n-1}$. After taking an action at iteration n, we observe a value $\hat{v}_{v,n}$ that is used to improve the estimate for $I^{post}_{v,n-1}$. We show how the ANN computes the statistical expectation $\bar{V}_{v,n-1}(I^{post}_{v,n-1})$. First, post-decision state $I^{post}_{v,n-1}$ maps to a contracted feature vector $\phi_v = (\phi_{v,f})_{f\in\mathcal{F}} \in \mathbb{R}^{|\mathcal{F}|}$, with each feature $f \in \mathcal{F}$ representing some property of the state. This way, we can estimate values for states that have not been observed previously. In addition, the state is typically both high-dimensional and sparse, which are undesirable properties for the input vector. At the core of the ANN is the layer of hidden nodes $\mathcal{Z}$, represented by a vector of functions $\sigma_v = (\sigma_{v,z})_{\forall z\in\mathcal{Z}}$ that transforms the input (i.e., the feature vector) to an expectation for the post-decision value. Both the inputs and outputs of σ_v are multiplied with weight vectors; we update these weights to fit the ANN to the observed values. We distinguish between input weights $w^{in}_{v,z,n-1} = (w^{in}_{v,z,f,n-1})_{\forall f\in\mathcal{F}}$ and output weights $w^{out}_{v,n-1} = (w^{out}_{v,z,n-1})_{\forall z\in\mathcal{Z}}$. Each hidden node takes the inner product of the input layer and the input weight vector corresponding to z and transforms it to an output value, i.e., $\sigma_{v,z} : (\langle w^{in}_{v,z,n-1}, \phi_v\rangle) \mapsto \mathbb{R}$. Thus, σ_v returns a vector of outputs. The expected value $\bar{V}_{v,n-1}(I^{post}_{v,n-1})$ is the inner product of this vector and the output weights, i.e., $\bar{V}_{v,n-1}(I^{post}_{v,n-1}) = \langle w^{out}_{v,n-1}, (\sigma_{v,z}(\langle w^{in}_{v,z,n-1}, \phi_v\rangle))_{\forall z\in\mathcal{Z}}\rangle$. After computing the estimated value, the ANN compares the discounted expected value $\gamma\bar{V}_{v,n-1}(I^{post}_{v,n-1})$ to the new observation $\hat{v}_{v,n}$. The approximation error at n is defined by

$$\delta_{v,n} = \frac{1}{2}\left(\gamma\bar{V}_{v,n-1}(I^{post}_{v,n-1}) - \hat{v}_{v,n}\right)^2 \quad . \tag{9}$$

We use a gradient descent algorithm to update the weights, with the gradient being defined as $\nabla_{v,n} = \left(\frac{\partial\sigma_{v,z}(\langle w^{in}_{v,z,n-1},\phi_v\rangle)}{\partial\langle w^{in}_{v,z,n-1},\phi_v\rangle}\langle w^{in}_{z,n-1},\phi_v\rangle\right)_{\forall z\in\mathcal{Z}}$. For this purpose we compute partial errors for each $\sigma_{v,z}$, defined by $\delta_{v,z,n} = \delta_{v,n}w^{out}_{v,z,n}$. Furthermore, let $\eta \in [0,1]$ be the learning rate determining how responsive the ANN is to observations deviating from the estimate. The weights are updated as follows:

$$\Delta w^{in}_{v,z,f,n} = \eta\delta_{v,z,n}\frac{\partial\sigma_{v,z}(\langle w^{in}_{v,z,n-1},\phi_v\rangle)}{\partial\langle w^{in}_{v,z,n-1},\phi_v\rangle}\langle w^{in}_{v,z,n-1},\phi_v\rangle\phi_{v,f} \qquad \forall(z,f)\in\mathcal{Z}\times\mathcal{F} \quad , \tag{10}$$

$$\Delta w^{out}_{v,z,n} = \eta\delta_{v,n}\frac{\partial\sigma_{v,z}(\langle w^{in}_{v,z,n-1},\phi_v\rangle)}{\partial\langle w^{in}_{v,z,n-1},\phi_v\rangle}\langle w^{in}_{v,z,n-1},\phi_v\rangle \qquad \sigma_{v,z}(\langle w^{in}_{v,z,n-1},\phi_v\rangle) \qquad \forall z\in\mathcal{Z} \quad . \tag{11}$$

4.3 Feature space

As pointed out in Section 4.2, we apply a contraction mapping on the state to obtain a more dense feature vector as input for the ANN. This mapping is described here. Based on the structure of the value function, the feature space designs tested on a related dispatching problem [2], and extensive preliminary experiments, we construct a feature space with the following features: (i) a constant, (ii) the cumulative job volume, (iii) the number of jobs, (iv) the number of jobs about to incur a lateness penalty, and (v) the number of jobs about to incur a failed job penalty.

5 Numerical experiments

This section discusses the policies used as benchmarks, the experimental design, and the numerical results. All algorithms are coded in C++ and run on a 64-bit Linux machine with a 4x1.60GHz CPU and 8GB memory.

5.1 Benchmark policies

Aside from comparing the centralized ADP policy (π^c) and decentralized ADP policy (π^{dc}) between themselves, we also compare them to four policies that we use as benchmarks. The first policy (π^1) selects a random action at each epoch and serves as a baseline policy. The second policy (π^2) always dispatches available jobs directly via the shortest path to their destination, thereby maximizing the time rewards. The second policy (π^3) dispatches jobs as late as possible (also using the shortest path), such that they arrive just in time at the destination. When a job must be dispatched due to urgency, all jobs held for transport via the same arc are dispatched as well, aiming to increase utilization of transport services. The fourth policy (π^4) estimates downstream costs corresponding to each action based on sampling future states and solving them with π^2. The seminal work of Bertsekas [9] refers to this widely accepted method as a rollout algorithm and is known to be competitive to value function approximation [11, 5].

5.2 Experimental design

We test instances of 4 different sizes, containing 3, 5, 10, and 25 agents. The vertices are generated randomly in a square area. The maximum degree of vertices varies with the instance size, e.g., the instance 5(3) indicates 5 vertices that each have up to 3 connected arcs. We randomly generate 10 replications of each instance, for which we run 500 iterations using π^c, π^{dc}, and the four benchmark policies. We set an instance-dependent total time limit to compute the ADP policies (in case of decentralized planning this is the cumulative time for all agents); the algorithm keeps performing learning iterations until the limit is reached. The time limits are selected to elicit the computational effects of instance size on both centralized and decentralized planning; in practice performance improvements are marginal after several thousands of iterations. Based on preliminary experiments, we set

		Centralized		Decentralized	
Instance	Time	# Iter.	$\lvert\mathcal{X}\rvert^{max}$	# Iter.	$\lvert\mathcal{X}\rvert^{max}$
3(2)	2h	62,039	1.98E^{11}	77,679	1.75E^{4}
5(3)	3h	2,143	4.02E^{27}	57,718	1.66E^{6}
10(4)	12h	3	7.18E^{73}	54,196	2.43E^{8}
25(5)	24h	0	5.71E^{186}	4,312	4.35E^{10}

Table 1: # iterations completed within designated time and the theoretical upper bound on joint action spaces per instance.

$\eta = 0.001$, $|\mathcal{Z}| = 5$ and $\epsilon = 0.01$. For $(\sigma_{v,z})_{\forall z \in \mathcal{Z}}$ we use so-called leaky rectified linear units [12]. The corresponding weights are initialized using the He initialization procedure [12].

The joint action space is described by $\mathcal{X} = \prod_{v \in \mathcal{V}} \mathcal{X}_v$ for centralized planning and $\mathcal{X} = \bigcup_{v \in \mathcal{V}} \mathcal{X}_v$ for decentralized planning. The computational burden for centralized planning increases exponentially with the number of agents and linearly for decentralized planning. However, in both cases the increase is exponential with respect to the number of adjacent arcs.

5.3 Numerical results

For both centralized and decentralized ADP, Table 1 shows the number of learning iterations completed within the shown run time and the theoretical upper bound of the joint action spaces. As the action space for centralized planning grows exponentially with respect to the number of agents, it becomes increasingly challenging to learn centralized policies for larger instances; for the instances with 10 and 25 agents no (meaningful) policy could be learned within time.

Figure 2 shows the average reward value achieved with each policy for all instances. The random policy π^1 yields the worst results and fails to generate positive rewards. Also the heuristic policies π^2 and π^3 are significantly outperformed by both ADP policies, as they do not consider downstream effects of decisions. The rollout algorithm π^4 is more competitive, but is still outperformed between 20% and 26% by the ADP policies. The rollout yields downstream value estimates of lower quality, thus resulting in less effective policies. For the instances 3(2), π^c performs 3.0% better than π^{dc}. For the 5(3) instances, π^c performs 3.2% worse than π^{dc}; note that much less iterations can be completed to learn π^c than for π^{dc}, negating the potential benefits of having global information. Centralized planning does not generate any meaningful policies for instances larger than 5(3) within reasonable time.

Figure 3 shows a breakdown of the performance of π^{dc} relative to π^c, with metrics >1 indicating an improvement. The metrics are the weighted reward, traveled transport distance, delivery time, and the number of successful (non-penalized) deliveries. Recall that these metrics are jointly embedded in the objective function and may be influenced by altering their corresponding weights. Although weighted rewards are quite similar, autonomous

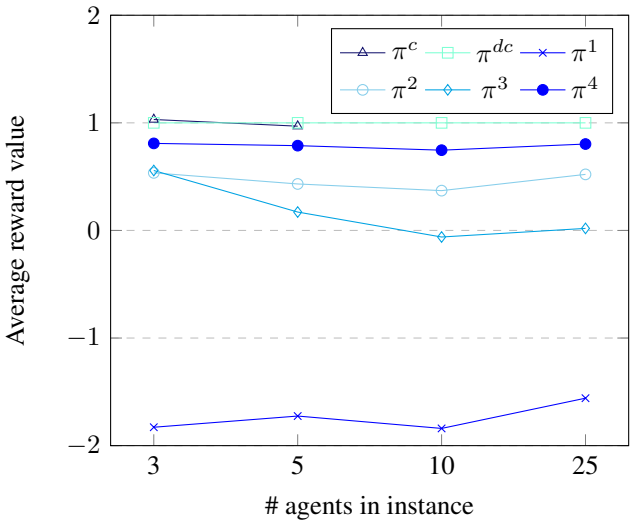

Figure 2: Performance per policy for each instance. Centralized policy π^c cannot be computed within time for the instances 10(4) and 25(5).

agents tend to dispatch jobs quicker and fail fewer jobs. The lack of information regarding jobs located elsewhere in the network may cause agents using decentralized planning to focus more on incurring time rewards and avoiding penalties. Finally, instances larger than 5(3) are not shown because – as mentioned earlier – π^c could not be computed.

6 Conclusions

This paper presents an ADP algorithm using neural network techniques to learn dispatch policies in flexible multi-segment transport networks. Four policies that we use as benchmarks are consistently outperformed by our algorithm. We compare the performance of policies that are learned either by a central planner that directs all transport flows in the network, or individually by autonomous carrier agents. The numerical results show that decentralized policies achieve a quality comparable to that of centralized policies, while requiring no information exchange between agents and a significantly lower computational effort. For larger instances, centralized planning fails to generate policies within reasonable time, whereas decentralized planning still generates high-quality solutions. As computational time grows linearly with respect to the number of agents, it is possible to apply decentralized planning to instances larger than tested in this paper.

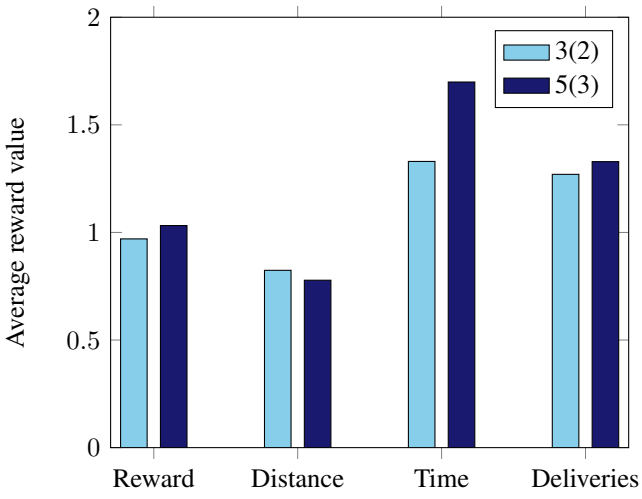

Figure 3: Performance of π^{dc} relative to π^{c}. Averages over all 3(2) and 5(3) instances respectively, metrics >1 indicate improvement.

References

[1] F. Ferrucci and S. Bock, "A general approach for controlling vehicle en-route diversions in dynamic vehicle routing problems," *Transportation Research Part B: Methodological*, vol. 77, pp. 76–87, 2015.

[2] W. J. A. Van Heeswijk, M. R. K. Mes, J. M. J. Schutten, and W. H. M. Zijm, "Freight consolidation in intermodal networks with reloads," *Flexible Services and Manufacturing Journal*, pp. 1–34, 2016.

[3] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013.

[4] A.-G. Lium, T. G. Crainic, and S. W. Wallace, "A study of demand stochasticity in service network design," *Transportation Science*, vol. 43, no. 2, pp. 144–157, 2009.

[5] W. J. A. Van Heeswijk, M. R. K. Mes, and J. M. J. Schutten, "The delivery dispatching problem with time windows for urban consolidation centers," *Transportation Science*, pp. 1–19, 2017.

[6] M. R. K. Mes, M. C. Van Der Heijden, and A. Van Harten, "Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems," *European Journal of Operational Research*, vol. 181, no. 1, pp. 59–75, 2007.

[7] H. Jung, B. Jeong, and C.-G. Lee, "An order quantity negotiation model for distributor-driven supply chains," *International Journal of Production Economics*, vol. 111, no. 1, pp. 147–158, 2008.

[8] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, 2011, vol. 2.

[9] D. P. Bertsekas, *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 2017, vol. 1, no. 4.

[10] S. S. Haykin, *Neural networks and learning machines*. Pearson Upper Saddle River, NJ, USA, 2009, vol. 3.

[11] M. W. Ulmer, D. C. Mattfeld, M. Hennig, and J. C. Goodson, "A rollout algorithm for vehicle routing with stochastic customer requests," in *Logistics Management*. Springer, 2016, pp. 217–227.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.