Coevolution and Transfer Learning in a Heterogeneous, Point-to-Point Fleet Coordination Problem

Logan Yliniemi
Oregon State University
Corvallis, OR, USA
logan.yliniemi@engr.orst.edu

Kagan Tumer
Oregon State University
Corvallis, OR, USA
kagan.tumer@oregonstate.edu

ABSTRACT

In this work we present a multiagent Fleet Coordination Problem (FCP). In this formulation, agents seek to minimize the fuel consumed to complete all deliveries while maintaining acceptable on-time delivery performance. Individual vehicles must both (i) bid on the rights to deliver a load of goods from origin to destination in a distributed, cooperative auction and (ii) choose the rate of travel between customer locations. We create two populations of adaptive agents, each to address one of these necessary functions. By training each agent population separate source domains, we use transfer learning to boost initial performance in the target FCP. This boost removes the need for 300 generations of agent training in the target FCP, though the source problem computation time was less than the computation time for 5 generations in the FCP.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithms, Management, Performance, Reliability

Keywords

Multiagent Learning, Transfer Learning, Coevolution, Logistics

1. INTRODUCTION

The use of semi tractor-trailers to move large amounts of goods from one place to another is the backbone of the economy in developed nations. In the United States, over 70% of all transportation of commercial goods was conducted by truck, dwarfing all other forms of freight transportation [2]. Trucking companies face the complex problem of routing their vehicles in such a way that they complete their contracted deliveries on-time, while spending minimal fuel and other resources. Fuel efficiency of individual vehicles has steadily increased in recent years, due to a high amount of research attention to aerodynamics and fuel efficient designs [8, 10, 27]. This, however, only addresses one side of the problem. If a company poorly assigns these delivery tasks to vehicles in its fleet, a large amount of fuel can wasted by vehicles traveling "empty miles"—miles traveled where a vehicle has no cargo [12]. Previous work on this subject centers around the Vehicle Routing Problem (VRP).

The VRP addresses the need to minimize the resources consumed in a road-vehicle-based logistics environment [6]. The original VRP creates a static customer set with a set of demands for a single good that must be satisfied by a single depot to the customer set. Each customer typically has a set window of time within which they will accept deliveries. Solutions to the VRP typically seek to minimize the number of vehicles necessary to complete the delivery set [6]. Algorithms that solve the classic VRP typically fall into one of three categories: route construction, local search, or metaheuristics [9]. Each of these solution types are typically nonadaptive and centralized, and require full observability of the system. The solutions also have very little generalizability, and solving a new problem instance will take as long as solving the first, even if the two are very similar [9].

We address this problem by framing the domain as a distributed multiagent system with adaptive agents. This puts our solution strategy in the category of metaheuristics, which are typically noted for being slow to calculate. However, we leverage the benefits of transfer learning, in which experience from one problem instance can be used to boost performance in another problem instance. By doing this, less adaptation is needed between problem instances, and acceptable performance can be attained in a new problem instance in significantly less time than solving the new system from scratch.

The classic VRP has been extended multiple times to include more realistic demands. One example includes soft time windows to the classic formulation, where a delivery may be made outside of the desired time, with a penalty assessed proportional to the time the delivery is outside of the desired window [9, 14]. Other extensions allow for the inclusion of multiple depots from which deliveries may originate [23, 15], backhauls that must be made in which the customers have goods to deliver to the depot [13], simultaneous pickups and deliveries for back hauls [7, 17], heterogeneous vehicles [25], and time-varying travel speed [28]. Each of these extensions incorporates an additional level of realism into the problem, but each also leaves out the contributions of the other problem formulations.

In collaboration with an industry partner, we identified key extensions to the VRP that best embody the day-to-day operation of a truckload or less-than-truckload carrier [16]. These extensions include (i) a fixed fleet size, (ii) soft time windows, (iii) heterogeneous vehicles, and (iv) multiple depots, which we expand such that each customer acts as a depot, creating a point-to-point delivery problem. We include *all* of these extensions in the domain considered in this work. We also include a novel extension of the VRP: (v) an elective, nonlinear tradeoff between travel time and fuel expenditures.

The tradeoff between travel time and travel cost serves to model that in many cases, there are multiple routes from one location to another that trade between time efficiency and fuel efficiency: very often a more fuel efficient route may be available, and simply take more time to traverse. The inclusion of these extensions to the Vehicle Routing Problem creates a problem domain that brings a different focus to the coordination required for a logistically-viable solution. Instead of a centralized controller that develops a set of routes, each agent must coordinate with the others such that the system performs well as a whole.

The crux of the problem then becomes controlling the vehicles in such a way that the fleet is well coordinated, to reduce "empty miles" and other wasteful fuel consumption. As such, we term our formulation of the VRP the Fleet Coordination Problem, or FCP.

While algorithms exist to address each of these extensions (i–v) individually, none have been created that incorporate them all. In this work we propose an adaptive, distributed control technique for assigning the responsibility for a given delivery event through a multiagent auction to minimize fuel consumption. By addressing the problem in this manner we can take advantage of transfer learning to allow experience gained in one problem instance to generalize to other "target" problem instances. This also allows us to train the agents initially on simple "source" problems to boost their adaptivity or performance in the target problem.

The major contributions of this work are to:

- Provide a distributed, adaptive solution strategy to a complex Fleet Coordination Problem
- Show that transfer learning allows agents trained in a simple source problem to significantly reduce required training time in the FCP
- Show that agents continue to learn through coevolution in the FCP
- Demonstrate robustness to calculation approximations and partial transfer learning

The remainder of this paper is organized as follows: Section 2 addresses background involving the VRP, transfer learning, evolutionary algorithms, and coevolution. Section 3 provides a complete description of the novel Fleet Coordination Problem addresssed in this work. Section 4 provides a treatment of the experimental methods and algorithms used in this work, including a full description of the source problems used. Section 5 contains the experimental results of this work, which show significant gains through using transfer learning in the FCP, even in the presence of calculation approximations or information loss. Section 6 draws conclusions from this work and addresses future research directions.

2. BACKGROUND

In Section 2.1, we define the Vehicle Routing Problem and describe a number of previously studied extensions to the VRP, as well as solution strategies that have been used to address these problems. In Section 2.2 we provide the relevant background on transfer learning and its previous use in various domains. Section 2.3 provides background on coevolution.

2.1 Vehicle Routing Problem

A classic version of the VRP is formalized as:

> **Definition : VRP** The classic Vehicle Routing Problem (VRP) consists of a depot D and a set of nV homogeneous vehicles $V = \{v_1, v_2, ..., v_{nV}\}$ with a common maximum load Q_{max} and maximum route length L_{max}. These vehicles must service a set of nC customers $C = \{c_1, c_2, ..., c_{nC}\}$ with demand $q_i \in \mathbb{N}$ for a good provided by depot D. Each customer must be serviced by exactly one vehicle, and each vehicle must return to the depot at the end of its route. The goal is to minimize the number of vehicles nV required to service all customer demand [6, 23].

Early approaches to the VRP included three primary methods. Direct solution approaches were only viable for small problems, while heuristic methods and methods based on the Traveling Salesman Problem were able to handle larger problem instances [6]. Since the development of these early solution strategies, work in the VRP has both focused on incorporating realistic extensions to the VRP as well as finding new solution strategies.

Two simple extensions include the Vehicle Routing Problem with Hard Time Windows and Vehicle Routing Problem with Soft Time Windows. These extensions add time bounds within which each customer may be serviced. In the case of hard time windows, no early or late deliveries are allowed. In the case of soft time windows, these deliveries are allowed, but are penalized proportionally to the deviation from the prescribed time window [9]. These variations increase the problem complexity significantly, and are readily incorporated into other VRP variations. Most VRP implementations use hard time windows as an implied constraint unless explicitly stated otherwise.

Two other extensions to the VRP include the Vehicle Routing Problem with Backhauling (VRPB) which provides each customer with a supply $s_i \in \mathbb{N}$ that must be hauled back to its originating depot, and the Vehicle Routing Problem with Simultaneous Pickups and Deliveries, which is a form of the VRPB in which the dropoff of the demanded goods and the pickup of the supplied goods must occur simultaneously (in order to minimize loading effort on the part of the customer) [13, 7]. These extensions are typically not incorporated into other variations on the VRP.

Another variant of the VRP is the use of heterogeneous, fixed-size fleets [25]. In this formulation, a fixed number of multiple types of vehicles is available to deliver goods to customers. Each different vehicle type has a unique maximum capacity and cost per unit distance traveled.

One final notable extension is the Vehicle Routing Problem with Multiple Depots, which changes the single depot D into a set of depots $D = \{d_1, d_2, ..., d_{nD}\}$. One notable treatment of this problem was carried out by Léauté et. al, who framed the problem as a Distributed Constrained Optimization Problem (DCOP) using various modern techniques to solve the DCOP such as SynchBB, DPOP, and P-DPOP before solving the resulting VRPs with a locally centralized controller [15].

In this work, we incorporate many of these into a single problem domain. The FCP, described in section 3, uses soft time windows, a heterogeneous fixed fleet, and as many depots as customers (creating a point-to-point delivery problem) as well as incorporating an extension not found in the literature: an elective tradeoff between travel speed and travel cost.

The solution strategies for various versions of the VRP have a number of shortcomings. Many of the solutions are not generalizable from one problem type to another (though some special circumstances do exist where solutions from one problem type can be generalized to another, e.g. solutions to the VRP with Soft Time Windows can be used for the VRP with Hard Time Windows if the penalty for early or late service is high enough), or even one problem instance to another of the same type [9]. Additionally, many of the algorithms are centralized, and require full system observability for calculations.

Decentralized solution strategies have been used to address some variations of the VRP [23, 15, 3]. Of these, however, some solve the problem using decentralized coordination, while others merely divide the problem into smaller VRPs and solve each of these with a centralized controller [15].

2.2 Transfer Learning

None of the solution approaches used for the various embodiments of the VRP would be suitable for the FCP without significant adjustment, or would only work on a specialized subset of problem instances within those allowed by the FCP. However, an adaptive, multiagent approach—wherein an agent senses information about its environment, reasons based on that information, and takes some action — bears a number of advantages for the FCP. First, framing the problem in a multiagent setting allows for an effectively decentralized approach with minimal communication between agents. Second, using adaptive agents can allow the use of transfer learning (TL) to leverage experience gained in one problem instance to increase performance in another problem instance [5].

At its simplest, transfer learning can allow an agent that takes actions based only on local state information to use the same policy in a different instance of an identically-formulated problems [26]. However, transfer learning can also be leveraged to use a simple training domain, or "source" to boost performance in a more complex problem domain, or "target", as long as the policies can be represented in a similar manner in both cases, and the experience gained in the source problem is valuable in the target problem.

That is, an agent trained on source problems similar to the target problem will gain benefits in performance or trainability in the target domain, but agents trained on a random task will not gain any performance benefits, and may in fact be hurt by the transfer [5, 18].

In this work we use a function approximator for each agent, in the form of a single-layer, feed-forward neural network. Such neural networks are powerful computational tools that can serve as function approximators and have been used in transfer learning in previous work [5, 21]. These neural networks have been used in applications as complex and diverse as computer vision, HIV therapy screening, and coronary artery disease diagnosis [1, 4, 22].

Success in transfer learning is typically a function of the similarity between the source problem and target problem, training time on the source problem, and validity of the agent's representation in both the source and target problems [11, 18]. In the ideal case, the agents representation fits both problems very well, and knowledge is well represented for transfer. This occurs when the states seen and actions taken are similar in both cases.

In this work we use transfer learning by producing single agent domains that act as a microcosm of the FCP, in which agents face similar, but not identical decisions. By training on these domains, described in Section 4 before directly using the developed policies into the FCP, we gain benefits both in initial performance and learning speed.

2.3 Evolution and Coevolution

To allow the agents to adapt to their environment, in hopes of increasing the performance of the system as a whole, we need a way to affect the policies with which the agents reason about which actions to take in whatever state they sense. In this work, we achieve this through the use of both evolutionary algorithms, and coevolution.

Evolutionary algorithms are a biologically-inspired computational technique in which a population of agent policies is first randomly generated, and then tested in some domain. After calculating a scalar measurement of an agent's "fitness" for each agent, those with lower fitness are replaced with slightly-altered copies of their higher-fitness counterparts. Through this random alteration and intelligent selection, system performance increases as the agents adapt to the domain to maximize their fitness calculation.

Coevolutionary algorithms leverage the concept of evolution for team-based domains. In coevolution, multiple separate populations are maintained, and are used in a shared simulation environment, where their fitness is evaluated based on how well they perform an assigned task as a member of a team made up of members from each population. An evolutionary algorithm is carried out on each population individually, such that the populations eventually produce agent policies that are well-suited in the team-based environment, to maximize the team's calculated fitness.

Coevolutionary algorithms have the potential to speed up a search through a complex space (which readily characterizes the FCP), but can often lead to a suboptimal area of the search space [19, 21]. This can be due to the agents learning to take a conservative strategy, being able to cooperate with a broader range of teammates [19, 20]. However, in this work we use an evolutionary algorithm on each agent population before using transfer learning and incorporating coevolution in the FCP (Section 4.2 – 4.3), so we have already guided the populations into favorable areas of the search space.

3. FLEET COORDINATION PROBLEM

The Fleet Coordination Problem is a variant on the classic VRP that includes the following extensions: (i) soft time windows, (ii) customer locations acting both as depots and delivery locations, (iii) a heterogeneous fleet of vehicles, (iv) a fixed fleet size, and (v) an agent-determined tradeoff between time efficiency and fuel efficiency (which can be intuitively thought of as the vehicle choosing a "speed" without much conceptual loss).

The FCP can be formally defined as:

Definition : FCP The Fleet Coordination Problem (FCP) consists of an allotted time $T = \{t | 0 \leq$

$t \leq T_{end}\}$, a set of customers $C = \{c_1, c_2, ..., c_{nC}\}$ in the 3-dimensional Euclidian space, a set of nP packages $P = \{p_1, p_2, ..., p_{nP}\}$, each consisting of a set of: a weight $0 \leq w_i \leq W_{max}$, a customer origin for the package $c_j^a \in C$ located at λ_j^a, a customer to deliver the package to, $c_j^b \in C$, located at λ_j^b, the beginning and end times within which the delivery must be completed, $t_j^a, t_j^b \in T$. Each package $p_j = \{w_j, c_j^a, c_j^b, t_j^a, t_j^b\}$, must be delivered point-to-point by one of a set of nV nonhomogeneous vehicles $V = \{v_1, v_2, ..., v_{nV}\}$ which are each described by fuel efficiency, weight, and allowed cargo weight $v_i = \{\eta_i, w_i, \kappa_i\}$. Each vehicle v_i travels along each of the nK "journey legs" described by edges connecting each customer directly to each other at a unitless rate of travel $R_k \in [0, 100]$ The goal is to maximize the system level utility $\mathbb{G}$, measured as a combination of the negative total fuel consumed by all members of the fleet and their on-time performance (Equation 10).

There are a few key points to note in this formulation. First, it is possible to travel from any customer c_a to any other customer c_b directly. This is an abstraction of a road system, which would realistically pass through a customer c_c if that customer was sufficiently close to the straight-line path. Each trip from a customer c_a to c_b consists of $nK = 10$ "journey legs" ("legs" for short), regardless of the length of these legs. Agents may decide on the tradeoff between fuel efficiency and time efficiency for each leg independently, but the decision holds for the entire leg. This assumption was made so that calculations would scale relative to the number of packages to be delivered in a system, rather than the distance travelled by the vehicles in an experimental run.

Also, the customers are not assumed to be on the Euclidian plane, and in fact exist in a three-dimensional environment. The slope between any two customer locations is limited to be less than a 6% grade. This adds the complexity of uphill and downhill travel into the decision-making required by each agent and the fuel efficiency calculation [24].

The fuel cost for traveling from customer c_a to customer c_b is characterized as a sum over all of the k legs of the journey:

$$F_{tot} = \sum_{k=1}^{nK} F_{elec,k} + F_{req,k} \tag{1}$$

where

$$F_{req,k} = \eta_i \alpha_1 \delta_k (1 + w_i \sin(S_k)) \tag{2}$$

$$F_{elec,k} = \eta_i \alpha_2 \delta_k R_{j,k}^2 + \eta_i \alpha_3 \delta_k R_{j,k} \tag{3}$$

where α_{1-3} are experimental coefficients that are held constant through experimental runs, η_i are vehicle-specific fuel-efficiency parameters, δ_k is the distance of leg k, w_j is the weight of truck j, S_k is the slope of leg k, and $R_{j,k}$ is the "rate-of-travel" of truck j over leg k [24]. This $R_{j,k}$ value can be loosely interpreted as a speed of travel but is more accurately described as both a function of speed and fuel efficiency of a vehicle's chosen route.

Intuitively, these fuel expenditure equations break down to this: Equation 2 calculates the minimum cost of travel between two points, which is a function of the distance between the two points, and the slope of the road between them. Equation 3 models that the vehicles may choose to move along more or less fuel efficient paths (modeled by the linear term) at a more or less fuel efficient speed (modeled by the quadratic term) [24].

Algorithm 1 FCP Execution Algorithm

```
for $j = 1 \rightarrow total\_packages$ do
  $\lambda_a \leftarrow$ Package origin location
  $\lambda_b \leftarrow$ Package destination
  for $i = 1 \rightarrow total\_vehicles$ do
    if vehicle $i$ is "busy" then
      Bidding agent $i$ bids $\beta_{i,j} = 0$
    else
      Bidding agent $i$ bids a value $\beta_{i,j} \in [0, 1]$
    end if
  end for
  Find highest bidder : $v_{win} = \underset{i}{\mathrm{argmax}}(\beta_{i,j})$
  Move $v_{win}$ to origin: Algorithm 2 $(\lambda_{v_i}, \lambda_a)$
  Increase weight of $v_{win}$ by package weight $w_j$
  Move $v_{win}$ to destination: Algorithm 2 $(\lambda_a, \lambda_b)$
  Decrease weight of $v_{win}$ by package weight $w_j$
  Determine if package $j$ was delivered within desired delivery window (Equation 6)
end for
```

The reason for the forms of these equations follows: for any journey, there is a physical absolute minimum fuel that has to be spent to complete the journey. This amount of fuel for journey k is F_{req}. Beyond that, due to the choice in driving habits of the driver, and route choices by a navigator, more fuel can be spent to get to a destination faster. This is represented by F_{elec}.

This fuel consumption model is not intended to compete with to the state of the art. This model was chosen to limit computation time, while still lending a degree of realism to the problem domain [16].

Paired with this, the rate-of-travel decisions $R_{j,k}$ also affect the time of travel between the origin and destination:

$$T_{tot} = \sum_{k=1}^{nK} \frac{\delta_k}{R_{j,k}} \tag{4}$$

where T_{tot} is the total time it takes for the vehicle to travel from the two locations, λ_a and λ_b.

It is also specifically important to note that all package deliveries must be completed; they may not be refused, and they stay in the domain until the completion of a delivery (some methods for internet routing allow for an amount of packet loss and design this into the approach; this is not viable for this problem domain).

Though we strove for applicability, This representation of the FCP is still a rough representation of reality, and cannot incorporate all facets of the real-world problem.

4. METHODS AND ALGORITHMS

We take a vehicle centric approach, wherein agents are associated with the vehicles; other agent-based distributed approaches are possible, with depots or packages themselves treated as agents, but in the FCP, choosing a vehicle-centric approach makes intuitive sense.

Because the target problem domain (the FCP, described in Section 3) is so complex, we break the agent responsibilities into two categories: (i) the movement of the agent from one location of another, and (ii) the decision of which agent will be responsible for each package. We explain these decisions in detail in Section 4.1. These two responsibilities are trained on different source tasks, which are laid out in Sections 4.2 and 4.3.

4.1 Distributed Auction and Agent Populations

First, an agent must decide how much the immediate importance the vehicle will give to speed and fuel efficiency, respectively, in order that the vehicle might be mobile across the domain. This "driver" decision must be made during each leg of each journey from a customer c_a to c_b, and this directly impacts both the fuel spent on that leg, and the time spent traversing that leg (Equations 1-4).

The other agent responsibility is choosing which package pickup and delivery events each vehicle will be responsible for. To solve this portion of the problem, we use a distributed, one-shot auction in which the highest bidder takes responsibility for traveling to the customer at which the package originates, picking up the package, and delivering it to its final destination. The agents each bid a value $\beta_i \in [0, 1]$ that represents how well-suited they believe their vehicle is for handling that package.

These two tasks, though they deal with similar information — an agent must consider the distance away from a package origin both in choosing a speed and in choosing a bidding value — are very different decisions. To address this, we took the split responsibilities and assigned them to two *completely separate* agent groups. That is, instead of one agent being assigned to each vehicle, a team of two agents, consisting of one "driving" agent and one "bidding" agent, is assigned to the vehicle, to work as a team. We characterize each of these agents as a 4-input, 10-hidden unit, 1-output feed-forward neural network.

The actions of each of these agents affects the performance of their teammate as well as the system-level performance $\mathbb{G}$, in turn. Because the actions taken by the two agents are very different, however, we choose to initially train them in different source environments, to leverage the maximum possible benefit from transfer learning. These two source problems are described in the following sections.

4.2 Evolution in Driver Source Domain

To train the driving agents, we pose the simplest possible problem, such that the driver learns the effect of its actions on the outcomes as quickly as possible. The source problem that we train the driver on is characterized as follows:

Definition : Driver Source Domain (DSD) A vehicle v is placed at the location of customer c_a, and assigned a package p_j of weight w_j. It must travel to the location of customer c_b (Algorithm 2), with the goals of minimizing total fuel consumed during the journey F_{tot} (Equation 1), and arriving before a time $T_f \in T$.

Algorithm 2 Travel Algorithm

```
Given origin and destination locations ($\lambda_a$, $\lambda_b$)
Determine the length of each journey leg, $\delta_k$
for $k = 1 \rightarrow journey\_legs$ do
    Select rate of travel $R_{(i,k)}$
    Calculate required and elective fuel spent (Equations 2 and 3)
    Calculate total fuel spent $F_{tot}$ (Equation 1)
    Calculate time spent $T_{tot}$ (Equation 4)
    Mark vehicle as "busy" for the time spent.
end for
```

Algorithm 3 Evolutionary Process

```
Initialize 100 population members of neural networks with small weights.
for $g = 1 \rightarrow end\_generation$ do

    Simulation Step (varies with domain)
    DSD: Simulate DSD $\forall$ agents $i$ (Section 4.2)
    or
    BSE: Calculate $\beta_i$ $\forall$ agents $i$ (Equation 7, Section 4.3)
    or
    FCP: Choose $nV$ agents at random, perform FCP (Algorithm 1); Repeat until all agents have participated once.

    Fitness Step
        Calculate fitness of all agents: $U_{i,g}$ $\forall$ $i$ (Equations 5, 8, 9 or 10)
        Sort agents from highest to lowest fitness

    Selection Step (select 20 survivors)
        set $counter = 20$
        set $survive_i = 0 \; \forall \; i$
    for $z = 1 \rightarrow 20$ do
        (select high-fitness survivors)
        With probability $(1 - \epsilon)$,
            $counter \leftarrow counter - 1$
            and set $survive_i = 1$
    end for
    for $z = 1 \rightarrow counter$ do
        (select random survivors)
        Select a random agent $j$ with $survive_j == 0$
        Set $survive_j = 1$
    end for

    Mutation Step (repopulate to 100 agents)
    for $Z = 1 \rightarrow total\_agents$ do
        if $survive_Z == 0$ then
            Select a parent network $Y$ where $survive_Y == 1$
            Set all weights of $Z \leftarrow$ weights of $Y$
            Mutate all weights using triangular distribution of mean 0, maximum and minimum change $\pm 0.05$
        end if
    end for

end for
```

With only two locations, one package, and one vehicle, this embodies an extremely simple training case. The single agent makes only ten decisions (the rate of travel for each leg of the journey) before receiving feedback, allowing for the feedback to be very specific to the individual agent and much easier to learn compared to the target multiagent, long-term environment.

Specifically, the agent is a single-layer, feed-forward neural network that takes as inputs the distance to the destination

$\delta_{(i,b)}$, the slope of the next leg of the journey S_k, a measure of "time pressure" ψ_j, and the vehicle weight w_v ; and gives as outputs $R_{j,k}$, the "rate-of-travel" of the vehicle it is controlling along the kth leg of the journey. ψ_j takes on a value of 1 if the vehicle can make it to its destination at a (unit-less) rate of travel of 75[1], a higher value if the vehicle is under more time pressure for the delivery (a faster speed is necessary), and a lower value if a lower speed would still result in an on-time delivery. This was done to give the agents a sense of time that scaled correctly with the problem. It is roughly equivalent to the human intuition of "being on time", while still en route to a destination.

The driving agents face exactly these same decisions within the FCP. Though the system-level effects of their decisions are not fully expressed within this domain, the simple principles that it is generally better to be on time than late, and better to be efficient about fuel expenditures are expressed within this domain. The specifics of the training algorithm follow.

The agents are trained through an evolutionary algorithm, in which a population of 100 agents is allowed to perform on the same instance of the DSD before downselection and mutation occurs (Algorithm 3). After each agent has performed once on a given problem instance (over the course of one generation), the problem instance changes; this allows the agent population to experience a wide variety of training situations to learn robust behaviors. In each generation, each agent is assigned a fitness for generation g based on fuel spent and whether they arrived before the prescribed time limit:

$$U_{j,g} = -F_{tot} - H(g)L(j) \tag{5}$$

where the fitness of agent j in generation g is $U_{j,g}$, the fuel consumed on the journey is F_{tot}, $H(g)$ is the positive "handicap" assessed for arriving late and is a function of the generation, and $L(j)$ is calculated as:

$$L(j) = max(0, T_{arrive} - T_f) \tag{6}$$

which returns zero if package j was on time, or the measure of time the package was late.

In Equation 5, the handicap $H(g)$ is initially zero, such that the agent has an opportunity to learn the function between its actions and the fuel efficiency of the vehicle over the journey. $H(g)$ then steadily increases as g increases, so that arriving on time becomes a higher and higher priority. As g nears the final generation gN, the fuel efficiency term and on-time term achieve a rough parity in terms of importance. Changes in the size of the experimental domain would necessitate an adjustment of the $H(g)$ function to maintain this parity.

The agents learn to achieve high performance on this task through an evolutionary algorithm, which mimics the process of biological evolution; high-achieving agents are very likely to survive, and lower-achieving agents are very likely to be replaced. In this implementation, we maintain a population of 100 agents, and allow 20 agents to pass directly to the next generation. The best-performing member of the population is always maintained, and 19 additional agents are selected; for each of the 20 agents with the best fitness values, the agent is selected to survive with probability $(1-\epsilon)$, and a random agent is selected to survive with probability $\epsilon = 0.3$. The value of ϵ was chosen to encourage slower population convergence to avoid local optima.

[1]75 was chosen as a "typical" rate of travel for this calculation, merely so that vehicles could "make up time" if necessary, at quadratic fuel cost. See Equation 3 for details.

The 20 surviving agents serve as parents for the 80 agents created for the next generation. Each new agent selects a parent agent from among the survivors, and after a step of mutation using a triangular distribution on each weight centered at zero, with maximum and minimum deviations of ± 0.05 in the neural network is, is entered into the population. This evolutionary algorithm process is outlined in Algorithm 3.

4.3 Evolution in Bidder Source Domain

In a similar manner, we must form a simple source domain to train the bidding agents. In our first iteration of this, we created a source similar to the driver source domain, which instead placed multiple vehicles near a pickup location and allowed the bidding agents to create bids for each vehicle. This led to unacceptable performance: for any given delivery, bidding agents learned to bid *relatively* higher for better suited vehicles, but they did not learn to create an appropriate *absolute* bidding gradient. In fact, the agents trained on this BSD only utilize about 1% of the available bidding space, that is, $\beta_{max} - \beta_{min} \leq 0.01$. This bidding strategy did not generalize well into the FCP.

Instead of this approach, we applied domain knowledge to create a supervised learning problem. Distance, time, vehicle weight and road conditions between the vehicle's current location and the pickup location all have an impact on a vehicle's suitability for a delivery. We can combine these in a linear fashion to create a target bid equation to train the population of agents. We call this equation the Bidder Source Equation, detailed below:

> **Definition : Bidder Source Equation (BSE)** We train agent i's bid for vehicle v (initially located at location λ_i) on package p_j using the equation:
>
> $$\beta_{train} = 1 - k_1\delta_{(i,a)} - k_2 S_{(i,a)} - k_3\psi_j - k_4 w_v \tag{7}$$
>
> where β_{train} is the bid, $\delta_{(i,a)}$ is the distance from the vehicle's current location to the package origin, $S_{(i,a)}$ is the average road slope between the vehicle's location and the package origin, ψ_j is a metric that characterizes the time available to make the delivery (see Section 4.2), and w_v is the weight of the vehicle. k_{1-4} are tunable parameters. Performance is not sensitive to these parameters, except that k_1 must be significantly larger than the rest.

With this equation as the source problem, we can create a random training instance and train directly on the error, in a case of supervised learning. That is, the fitness U_i of an agent i is:

$$U_i = -|\beta_i - \beta_{train}| \tag{8}$$

We allow each agent to evaluate the circumstances provided by the problem instance into a bid, and then use the same evolutionary algorithm outlined in Section 4.2 in Algorithm 3. By using the BSE, we attained bidder behaviors that used a much larger portion of the available bidding space ($\beta_{max} - \beta_{min} \approx 0.9$). This bidding range was much more appropriate to transfer into the FCP.

The justification for the form of Equation 7 is as follows. Each term in the linear combination expresses a simple fact:

for example the first term states that a vehicle closer is more suited for a delivery than one further away; the last states that a lighter vehicle is more suited than one that is heavier. The linear combination of these factors is the simplest possible form that expresses these facts. By allowing our bidders to train on this equation, we improve their performance over a random neural network. This equation does not seek to represent an ideal bidding formulation for the FCP, it merely acts as a starting point, from which the agents may deviate as the evolutionary process continues.

4.4 Coevolution in Target Domain

The agent populations, trained first in their source domains (the DSD and BSE), are then put into use in the target domain, the FCP. We choose to keep the two populations of agents separated, calculating their fitness with the same metric in each experiment, but allowing them to be evolved separately. By allowing this, and by randomly pairing the agents together in each problem instance together, we employ the concept of coevolution.

In the experiments conducted for this work, a set of 10 trucks is assigned to service 25 customers for a series of 1000 package delivery events. For each generation, we draw 10 agents from each population that have not yet been used in the current generation, pair them randomly, and assign the teams of agents associated with vehicle i fitness values based either on their local utility $\mathbb{L}_i$, or the global system utility $\mathbb{G}$. We calculate $\mathbb{L}_i$ as the negative sum of all fuel spent by that vehicle, plus a positive term for each package delivered on-time.

$$\mathbb{L}_i = \sum_{j=1}^{nP} I_i(j)[-F_{(tot,j)} - HL(j) + \phi] \tag{9}$$

where nP is the total number of packages in the problem instance, $I_i(j)$ is a function that indicates whether vehicle i was the maximum bid for package j, $F_{(tot,j)}$ is the total fuel expended delivering package j, H is the constant handicap coefficient for a late delivery, $L(j)$ is calculated by Equation 6, and ϕ is a constant "bonus" for making a delivery.

Because fuel costs are always a negative value, without the positive bonus for delivering a package, the agents quickly learn to bid low so that another agent might have to make the delivery, making the first agent earn zero fitness for that delivery event, while the other incurs the negative cost. This increases the fitness of the first agent at a cost to the system level performance. Conversely, if ϕ is set too high, all agents bid very high for every delivery, because the fuel costs incurred are dwarfed by the bonus received for completing the delivery.

We calculate the global system utility $\mathbb{G}$ as the sum of all local rewards, without the bonuses (ϕ):

$$\mathbb{G} = \sum_{i=1}^{nV} [\mathbb{L}_i] - (nP)\phi \tag{10}$$

At the system level we are concerned with the sum total of fuel spent; the package delivery bonuses merely add a constant to the global reward, which does not affect learning. Because all of the agents are scored on the overall fuel consumption, they learn not shy from taking a package that they are well-suited to deliver, making the bonus term unnecessary. The on-time delivery term remains, because this is a matter of concern to a truckload-hauling carrier: having a high on-time-percentage can be a selling point in attaining new contracts, and by changing the H term in Equation 9, we can potentially cause the agents to make many hard-time deliveries (high H) or many soft-time deliveries (H low). This $\mathbb{G}$ term will always evaluate to be negative; the agents strive to maximize the system utility, thus minimizing the fuel spent to make a delivery on time.

4.5 Approximation and Partial Transfer

In order to show that our approach to solving the FCP is not brittle to changes in experimental parameters or small changes in procedure, we choose to demonstrate results on two additional forms of complication. First, we pose a situation in which none of the drivers are trained in any source problem, while bidders are trained in the BSE. In the target problem, coevolution is allowed to proceed as normal.

Additionally, to demonstrate that a variety of function approximators could be suitable for this solution strategy, we replace the sigmoid function in the neural network we used with a gross approximation which requires far less computational power: a 3-piece linear function:

$$f(x) = \begin{cases} 0 & : x < -2 \\ 0.25x + 0.5 & : -2 \leq x \leq 2 \\ 1 & : x > 2 \end{cases} \tag{11}$$

The training of the network and weights remained the same, but for the entire training process from source to target, the piecewise linear function was used instead of the sigmoid. Note that though this function is not differentiable, we do not use a gradient-based approach in this work; evolution still functions using this approximation by mutating the weights associated with the networks and evaluating fitness.

5. RESULTS AND DISCUSSION

In this work, we used all 8 methods (detailed below) on a series of 30 randomly-generated instances of the FCP, with a fleet of 10 vehicles serving 25 customer locations, over a course of 1000 package deliveries. In training the bidders, we used values of $\{k_1, k_2, k_3, k_4\} = \{0.6, 0.1, 0.1, 0.2\}$. We created a timescale long enough that the package congestion would be low, keeping the problem instance difficulty on the low end of those available through the FCP, and allowing all deliveries to be completed while limiting vehicles to carrying out one delivery at a time. We chose to use different problem instances for our statistical trials to show the robustness in our methods and results. For each statistical run, the same problem instance was used for all methods test, and reported global system utility $\mathbb{G}$ was normalized with respect to the mean of the first generation of all trials, except full transfer learning. This was done to allow comparison across statistical trials, and to emphasize that the global system utility $\mathbb{G}$ is a *unitless* quantity that represents a combination of the fleet's fuel consumption and on-time performance (Equation 10). Error is reported as the standard deviation of the mean,[2] and in many cases is smaller than the symbols used to plot. To validate our distributed approach to the novel FCP presented in Section 3, we desired to compare the effect of using a local fitness evaluation ($U_i = \mathbb{L}_i$) and a global fitness calculation ($U_i = \mathbb{G}$) for

[2]The deviation of the mean for N statistical runs is calculated as $\frac{\sigma}{\sqrt{N}}$

coevolution in the FCP, and testing these baseline measurements against the use of full transfer learning, using both DSD and BSE source environments. Additionally, we wish to demonstrate that our approach is not brittle to lost information or approximations, and compare these results to the baseline cases, creating a set of eight experimental methods. We show results in the following scenarios:

1. No transfer learning with fitness calculated through local and global utilities (Section 5.2)
2. Full transfer learning with global and local fitness during coevolution (Section 5.2)
3. Partial transfer learning (Section 5.3)
4. Transfer learning using a linear approximation of a sigmoid (Section 5.4)

5.1 Learning From Scratch

First, as a validation of our approach and methodology for the FCP, we compared the results of training the agents in the target FCP from scratch, with no transfer learning, against the use of full transfer learning from both source problems, as outlined in Section 4. Figure 1 shows these results, which show a substantial gain in system performance over the training period, regardless of whether the local or global training signal was used. This is because of the strong coupling between the two calculations, as the global utility $\mathbb{G}$ is formulated as a sum of local rewards $\mathbb{L}_i$, with a constant term subtracted (Equation 10).

5.2 Full Transfer Learning

We note that the full transfer learning case shown in Figure 2 thoroughly outperforms learning from scratch, with initial performance that is within 10% of the best converged performance of any algorithm tested. It is important to note here that the computation time for the full transfer learning case is roughly the same as 5 additional generations of training time in the FCP (as the source problems are much simpler). Full transfer learning, using both the DSD and BSE source domains to train the driver and bidder agents, is extremely effective in the FCP, and results of comparable quality can be obtained in only 10% of the training time required, when compared to learning from scratch. All of these results hold true whether the local utility $\mathbb{L}_i$ or global utility $\mathbb{G}$ is used for the fitness evaluation calculation, as the resulting performance is nearly identical.

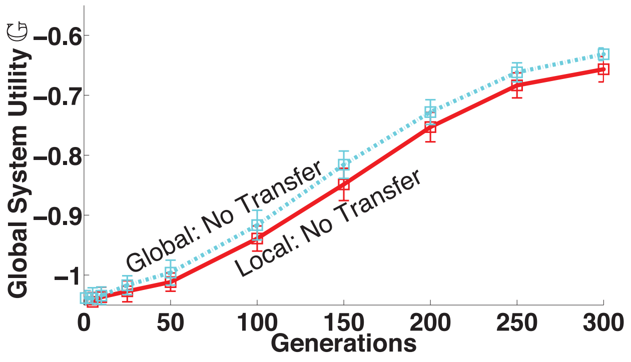

Figure 1: Comparison of agents evolved in the FCP using as their fitness evaluation: $\mathbb{G}$, the global system utility (upper, dotted line); $\mathbb{L}_i$ (lower, solid line). Note that in 300 generations, both fitness calculations lead to improvement in system performance.

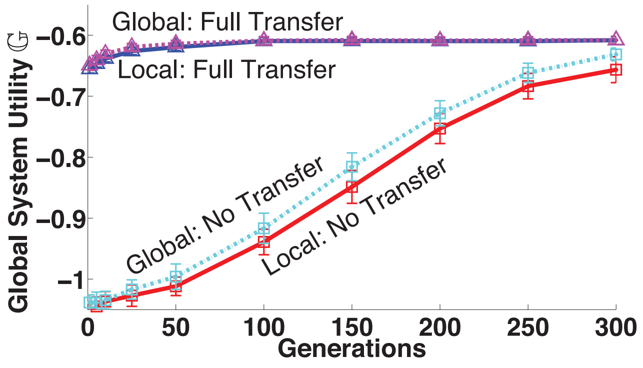

Figure 2: Performance in the FCP of agents coevolved solely in the FCP (lower lines with square markers) against agents evolved in separate source domains transferred into the FCP (upper lines with triangle markers). Note that the two full-transfer learning cases overlap significantly, and the boost in performance persists over 300 generations. Approximate computing time for the source problems in the full transfer cases was approximately 5 generations on this scale.

5.3 Partial Transfer Learning

We also show that our methodology in this work is robust to potential failures. First, we pose a scenario in which we allow the bidders to be trained on their source problem, but force the driving agents to learn from scratch in the FCP target environment: only one of the two agent populations benefits from transfer learning.

As seen in Figure 3 we still see some benefits both in initial performance ($\sim$10%) and in learning speed in both the local and global reward cases. Because of the simpler mapping from actions taken to fitness calculation (the bidders are taking reasonable actions, instead of random ones as they would when learning from scratch), noise is effectively removed from the fitness calculation and the rate of performance increase is improved. The driving agents are able to learn to take reasonable actions before the bidding agents diverge from making good decisions because of the reward signals received in the FCP. It is important to note that in all of these cases, the driving agents and bidding agents are always receiving the same reward.

When we perform the same experiment in reverse, allowing the driving agents to use transfer learning, while forcing the bidding agents to start learning from scratch, we see performance that is almost at the level of full transfer learning: in the tested instances of the FCP, the drivers are capable of wasting far more fuel than poorly assigned bids. This is because the worst effect a poor bid can have is a vehicle traversing across the experimental domain, while in this formulation a driver is allowed to choose an excessive speed that wastes significantly more fuel. These results were omitted from Figure 3 for readability. In higher package-congestion

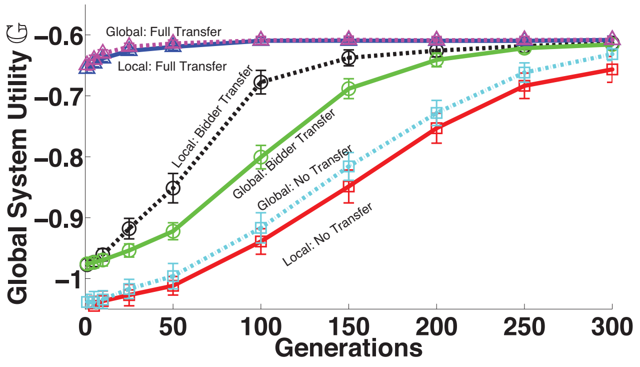

Figure 3: Experimental results in the FCP using three forms of transfer learning: no transfer (bottom, blue and red), full transfer (top, triangles), and bidder-only transfer learning (middle, green and black) which could correspond to an information loss scenario. Note that the full transfer learning cases overlap each other.

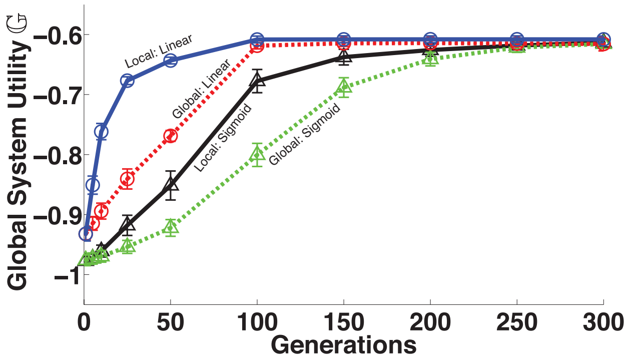

Figure 4: Experimental results in the FCP using bidder transfer learning and coevolving on the global system utility $\mathbb{G}$ or local agent utility $\mathbb{L}_i$. Because of the relatively simple problem instance, a neural network using a linear function as an approximation of the sigmoid itself actually outperformed the full-accuracy neural network.

cases or cases in which the system designer limits the speed at which the vehicles may travel to a greater degree, we expect that the bidding agents would have a stronger contribution to overall system performance.

5.4 Transfer Learning with Linear Approximations

Finally, we examine whether the neural network function approximation (which, with its many embedded sigmoids, can be very computationally expensive) is strictly necessary for our methods to work in the FCP domain. We replace the sigmoid function in each of these calculations with a 3-piece linear function with the same slope as the sigmoid at zero, seen in Equation 11.

In Figure 4, we show that using the linear approximation of the sigmoid in the FCP with bidder transfer learning actually leads to better system performance than using the neural network itself, converging to similar performance more than 100 generations faster. Here, it is important to note that the neural network using sigmoids itself is a function approximator, and the linear piecewise functions simplify this approximation.

Because the particular problem instance of the VRP shown is low-congestion enough, the linear function is able to embody all necessary information for treatment of this problem. In more congested problem instances, we expect that this would not be the case, and that the neural network using sigmoids would begin to outperform the linear approximation at some level of problem complexity or congestion.

6. CONCLUSION

In conclusion, in this work we have proposed the novel combination of VRP variations into the Fleet Coordination Problem domain. We proposed an adaptive solution strategy that leverages benefits from the fields of multiagent systems for decentralization, neural networks for function approximation, neuro-evolution for agent training, transfer learning for boosting initial performance and maintaining policy applicability over problem instances, and coevolution for simplifying the agent responsibilities and maintaining the ability for agents to retain their skills in addressing these different responsibilities in the FCP.

Though we trained on two simple source domains before placing agents in the FCP target domain, even after coevolving the agents on the combined FCP, we can still transfer this experience through a policy transfer, by maintaining the agents' policies and only changing the problem instance. Our experimental methods suggest that we can very easily take agent populations trained in one FCP and transfer their knowledge to an FCP of similar complexity with success, and work in the near future includes transferring agent experience from a simple FCP to a more complex, congested FCP instance. The FCP parameters for vehicles, packages, time, and customers in this work were chosen such that a suitable solution could definitely be found; problem difficulty can be increased by decreasing available resources to work with, or increasing demand. We expect that this "stair step" method of training agents may provide better performance in complex FCP instances than transferring straight from the original source problem to the final target problem, or learning from scratch. We seek to discover if there is a problem instance that cannot be successfully learned from scratch, which is achievable through the use of transfer learning.

Additionally, we wish to frame the FCP as a multi-objective problem, and incorporate multi-objective metrics into our treatment of the problem, including fitness shaping techniques to assist the agents in discerning their particular contribution to the system as a whole.

7. REFERENCES

[1] Amr Ahmed, Kai Yu, Wei Xu, Yihong Gong, and Eric Xing. Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks. *ECCV*, pages 69–82, 2008.

[2] Felix Ammah-Tagoe. Freight in america: 2006. Technical report, U.S. Department of Transportation, Washington D.C., January 2006.

[3] K. Savla Arsie, A. and E. Frazzoli. Efficient routing algorithms for multiple vehicles with no explicit

communications. *IEEE Transactions on Automatic Control*, 10(54):2302– 231, 2009.

[4] Steffen Bickel, Jasmina Bogojeska, Thomas Lengauer, and Tobias Scheffer. Multi-task learning for hiv therapy screening. *ICML*, pages 56–63, 2008.

[5] Rich Caruana. *Multitask Learning*. PhD thesis, Carnegie Mellon University, September 1997.

[6] Nicos Christofides. The vehicle routing problem. *Revue Francaise d'Automatique, d'Informatique et de Recherche Operationelle*, 10(2):55–70, February 1976.

[7] J Dethloff. Relation between vehicle routing problems: an insertion heuristic for the vehicle routing problem with simultaneous delivery and pick-up applied to the vehicle routing problem with backhauls. *Journal of the Operational Research Society*, 53:115–118, 2002.

[8] Richard Henry Distel and Richard Albert Distel. Apparatus to improve the aerodynamics, fuel economy, docking and handling of heavy trucks. Patent US 7,748,771 B2, Distel Tool and Machine Company, 2010.

[9] Miguel Andres Figliozzi. An iterative route construction and improvement algorithm for the vehicle routing problem with soft time windows. *Transportation Research Part C: Emerging Technologies*, 18(5):668–679, October 2010.

[10] Dario Guariento. System for completely closing the space between the cab and semi-trailer of an industrial or commercial vehicle, to improve the aerodynamics of the vehicle. Patent EP 1 870 320 A2, IVECO, 2007.

[11] Steven Michael Gutstein. *Transfer Learning Techniques for Deep Neural Nets*. PhD thesis, University of Texas at El Paso, May 2010.

[12] J Hine, A Barton, C Guojing, and W Wenlong. The scope for improving the efficiency of road freight transport in china. In *7th World Conference on Transport Research*, 1995.

[13] Charlotte Jacobs-Blecha and Marc Goetschalckx. The vehicle routing problem with backhauls: Properties and solution algorithms. Materials Handling Research Centre Technical Report MHRC-TR-88-13, Georgia Institute of Technology, Atlanta, 1993.

[14] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 1992.

[15] Thomas Léauté, Brammert Ottens, and Boi Faltings. Ensuring Privacy through Distributed Computation in Multiple-Depot Vehicle Routing Problems. In *Proceedings of the ECAI'10 Workshop on Artificial Intelligence and Logistics (AILog'10)*, 2010.

[16] Daimler Trucks NA. Personal communication.

[17] Gábor Nagy and Said Salhi. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European Journal of Operational Research*, 162:126–141, 2005.

[18] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010.

[19] Liviu Panait. Theoretical convergence guarantees for cooperative coevolutionary algorithms. *Evolutionary Computation*, 18(4):581–615, 2010.

[20] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Journal of Autonomous Agents and Multi-Agent Systems*, 11:387–434, 2005.

[21] Padmini Rajagopalan, Aditya Rawal, and Risto Miikkulainen. Emergence of competitive and cooperative behavior using coevolution. *GECCO*, pages 1073–1074, 2010.

[22] Daniel L. Silver and Robert E. Mercer. Sequential inductive transfer for coronary artery disease diagnosis. *International Joint Conference on Neural Networks*, 2007.

[23] Andrei Soeanu, Sujoy Ray, Mourad Debbabi, Jean Berger, Abdeslem Boukhtouta, and Ahmed Ghanmi. A decentralized heuristic for multi-depot split-delivery vehicle routing problem. *IEEE International Conference on Automation and Logistics*, 2011.

[24] Gwang-Geong Soung. Method and device for measuring slope of driving road. Patent 5,703,776, Hyundai Motor Company, Ltd., 1995.

[25] C.D. Tarantilis, C.T. Kiranoudis, and V.S. Vassiliadis. A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. *European Journal of Operational Research*, 152:148–158, 2004.

[26] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? *Advances in Neural Information Processing Systems*, pages 640–646, 1996.

[27] William Burley Uland. Under-vehicle aerodynamic efficiency improvement device. Patent Application Publication US 2005/0146161 A1, 2005.

[28] Yang-ByungPark and Sung-HunSong. Vehicle scheduling problems with time-varying speed. *Computers in Industrial Engineering*, 33(3-4):853–856, 1997.