AN ABSTRACT OF THE THESIS OF

Logan Michael Yliniemi for the degree of Master of Science in
Mechanical Engineering presented on April 23, 2012.

Title:

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

Abstract approved: _____

Kagan Tumer

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 in 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.

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

by

Logan Michael Yliniemi

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented April 23, 2012
Commencement June 2012

Master of Science thesis of Logan Michael Yliniemi presented on April 23, 2012.

APPROVED:

Major Professor, representing Mechanical Engineering

Head of the School of Mechanical, Industrial, and Manufacturing Engineering

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Logan Michael Yliniemi, Author

ACKNOWLEDGEMENTS

I would like to acknowledge and thank my advisor, Dr. Kagan Tumer, for supporting my research and providing guidance in pursuit of this work.

I would also like to acknowledge Cristin Paun and Derek Rotz of Daimler Trucks North America for their input.

Additionally, I would like to thank my friends and colleagues in the AADI lab for their support, criticisms, and demeanour throughout this process.

A special thanks also to Dr. Joseph K. Davidson at Arizona State, for starting me down this path.

Finally, I would like to thank Carey E. and Elizabeth P. Yliniemi for their support and guidance through the years.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

DEDICATION

This work is dedicated in memory of Marion Perko.

"Mind your nickels, that your dollars mind themselves."

Chapter 1 – Introduction

> "The general who wins the battle makes many calculations in his temple before the battle is fought. The general who loses makes but few calculations beforehand."
> - *Sun Tzu*

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 consuming 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 [19, 24, 62]. This, however, only addresses one side of the problem, and ignores any interactions that could occur because of the delivery task assignment process. If a company poorly assigns these delivery tasks to vehicles in its fleet, a large amount of fuel can be wasted by vehicles traveling "empty miles"—miles traveled where a vehicle has no cargo [27]. Even though each vehicle may consume less fuel per mile, the fleet as a whole could be wasting large amounts of fuel on unnecessary travel. 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 [14]. 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 [14], though other metrics such as cost of making the required deliveries have been used as well [10].

The VRP, in the original formulation, is a clear extension of the Travelling Salesman Problem [36], a well known NP-hard combinatorial optimization problem. The VRP reformulates the problem in such a way that it is appropriate for study within the realm of logistics, but in keeping close ties to its predecessor, the original VRP formulation leaves out an incredible amount of real-world applicability concerns. In addition, the scope of the problem is so narrow that it only applies to a very small subset of real-world logistics problems.

The VRP assumes a single depot that supplies a single type of goods to a static set of customers. This situation is ignorant of reality. In a more realistic case, the depot would both be a source for deliveries to a changing set of customers, as well as a destination for deliveries from primary suppliers, and may handle multiple types of goods. Additionally, with the advent of third-party logistics companies, many times there is no singular "depot", and instead there are just a series of customers with demand for some goods that must be delivered and a supply of others that must be taken to another customer. The combinations and possibilities

here are as endless as the real world itself.

Even ignoring the different circumstances that may change the VRP, there are a number of considerations that completely escape the scope of the problem formulation. The problem typically takes place assuming flat, trafficless roads with no required stops. The real world does not resemble this at all; it is a tangled mess of roadways with toll booths, weighing stations, police officers, reckless drivers and accidents, and even farm equipment sharing the roadways. Drivers must sleep, trucks must be refueled or undergo maintenance. Rain, snow, sleet, hail, and the odd strike or protest can additionally affect the real progress of a logistics carrier. It is necessary that much of this must be ignored in order to create a tractable problem that one might address using engineering techniques.

The original formulation of the VRP offers an NP-hard combinatorial optimization problem [26, 28] that has spawned decades of research, in which many different solution methods have been developed. The amount of effort directed at the problem speaks to its central importance within the realm of managing a logistics or distribution system.

Algorithms that solve the classic VRP typically fall into one of three categories: route construction, local search, or metaheuristics [20]. 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 [20].

In addition to the development of various different types of solution meth-

ods, 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 [20, 33]. Other extensions allow for the inclusion of multiple depots from which deliveries may originate [35, 51], backhauls that must be made in which the customers have goods to deliver to the depot [30], simultaneous pickups and deliveries for back hauls [18, 39], heterogeneous vehicles [55], and time-varying travel speed [44]. Each of these extensions incorporates an additional level of realism into the problem, but by treating them individually, each also leaves out the contributions of the other problem formulations, as well as any of the other concerns that might contribute to a more realistic problem formulation.

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 [38]. 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 four* of these extensions in the domain considered in this work. We also include a novel extension of the VRP: (v) an elective, non-linear 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.

These five extensions have each been studied individually—with the exception of (v)—and solution strategies have been identified and developed. However, the solutions for each of these problems individually are unfit for application to the additional complications offered by the inclusion of the other extensions. By incorporating all five extensions into a single realization of the VRP, which we simply term the Fleet Coordination Problem—FCP—we create a domain that seeks to embody the day-to-day operations of a truckload or less-than-truckload carrier, but as a tradeoff, it is quite complex, and we have to use a number of solution techniques to overcome this complexity.

Traditional optimization solutions like route construction or local optimization, while attaining favorable results in many variations of the VRP, have a number of significant challenges to overcome in this domain. Attempting to optimize globally leads to a number of parameters and possible solutions that is extremely large, and the types of heuristics that might be used would need to be developed to augment the heuristics commonly in use today to accomodate the additional complexity.

Stochastic search methods and metaheuristics suffer from similar problems, though it is possible to distribute the problem such that only local information is needed for the stochastic search to take place. However, the problem in either can case become intractable because of either too large of a search space (if global information is used) or a lack of big-picture understanding about how the local decisions affected the global performance, which are concerns that need to be faced.

To address these issues, we first applied a distributed, adaptive, agent-population-based learning approach to the problem. Even this proved to have significant problems in attaining acceptable performance, so we used two techniques to make the problem tractable: transfer learning, and problem decomposition to allow coevolution.

At the most basic, transfer learning is a technique that can take advantage of the presence of similarities between multiple instances of the same problem domain to boost performance in a previously unseen problem instance. This was succinctly demonstrated within [11], wherein an autonomous program drove a virtual race car through a series of tracks. When presented with a track that it had not seen before, it was still capable of completing the track successfully.

Transfer learning does not have to be limited to instances of the same task, however, and can also be used to generalize across different tasks. This has been done in sub-tasks related to Robot Soccer, wherein agents trained on one training task performed better on a related task than one with no previous experience. In some cases there were no significant differences between the tasks, while in other cases, a significant step up in task difficulty was seen [60].

Transfer from one domain to a different domain has also been utilized, to great effect. In 2007, Taylor and Stone showed that transfer learning is possible amongst domains that are as loosely connected as being games in which a physical position is the most important governing factor [56].

Taylor, Whiteson and Stone also showed in 2006 that it is possible to transfer from domains of different complexity levels using neural networks [58]. They

successfully used a policy trained on 3v2 agent keepaway for use in a 4v3 agent keepaway setting, after minimal modifications. This training cut more than half of the required training time to reach a predetermined benchmark level of performance in some cases.

It is possible to use transfer learning for even broader uses than simply scaling up a domain. In [57], Taylor and Stone note: "Sutton et al. (2007) motivate this approach to transfer by suggesting that a single large task may be most appropriately tackled as a sequential series of subtasks." This is exactly the approach that we take; we decompose the target FCP into a series of subtasks and learn those tasks first before tacklink the larger FCP.

By decomposing the responsibilities of an individual agent, we allow these responsibilities to be developed in separate source environments, allowing the agents to learn both what types of deliveries they should bid highly on as well as the rate of travel they should adopt between locations. By learning these two responsibilities on an individual level, the fleet learns to coordinate as a whole such that the number of "empty miles" and otherwise wasted fuel is reduced.

The major contributions of this work are to:

- Incorporate multiple distinct extensions of the Vehicle Routing Problem into a complex Fleet Coordination Problem
- Provide a distributed, adaptive solution strategy to the Fleet Coordination Problem
- Show that transfer learning allows agents trained sequentially in separate

simple source problems to reduce required training time in the FCP by an order of magnitude

- Show that agents continue to learn through coevolution in the FCP
- Demonstrate robustness to calculation approximations and partial loss of transferred information
- Show robustness to a simulated traffic distribution that both increases fuel expenditures and decreases rate of travel
- Show that our solution strategy produces near-optimal solutions in a simple case where the optimal is calculable

The remainder of this work is organized as follows: Chapter 2 addresses background involving the VRP, transfer learning, evolutionary algorithms, and coevolution. Chapter 3 provides a complete description of the novel Fleet Coordination Problem addresssed in this work. Chapter 4 provides a treatment of the experimental methods and algorithms used in this work, including a full description of the source problems used. Chapter 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. Chapter 6 draws conclusions from this work and addresses future research directions.

Chapter 2 – Background

> "Thus, what enables the wise sovereign and the good general to strike and conquer, and achieve things beyond the reach of ordinary men, is foreknowledge." - *Sun Tzu*

In Section 2.1, we define the Vehicle Routing Problem and describe a number of previously studied extensions to the VRP. In Section 2.2–2.4, we describe the common categories of solution strategies for the VRP and some specific examples. In Section 2.5 we provide the relevant background on transfer learning and its previous use in various domains. Section 2.6 provides background on coevolution.

2.1 Vehicle Routing Problem

A classic version of the VRP is formalized as:

> **Definition 1 : 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 [14, 51].

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 [14]. 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 [20]. 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) [18,

30]. 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 [55]. 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 [35].

In this work, we incorporate many of these extensions into a single problem domain. The FCP 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 FCP is detailed further in Chapter 3.

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 [20]. 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 [3,35,51]. 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 localized controller [35].

Possibly the most useful categorization of solution types to the FCP breaks the solutions developed into three different categories:

1. Construction Heuristics
2. Local Search Heuristics
3. Metaheuristics

Construction heuristics simply follow a set of rules to construct the routes the vehicles will take; local search heuristics typically alter constructed routes; metaheuristics are a broad category of solution strategies that are typically more computationally demanding. These are discussed in turn in the following sections.

2.2 Construction Heuristics

Construction Heuristics solve the VRP by building partial routes, one customer at a time, and assembling these together to form routes. Possibly the best known of these heuristics was developed by Clarke and Wright [15]. In this, the broad

problem is first broken up into clusters, and then the Travelling Salesman Problem is performed on each set of points; with enough clusters, the optimal TSP solutions can be found. Expanding upon this work, similar techniques were used by Solomon [52].

Construction heuristics typically fall into either the broad umbrella of parallel or sequential heuristics. In the parallel case, multiple routes are constructed simultaneously, where the sequential heuristics limit themselves to constructing one route at a time. Most modern treatments of the VRP do not limit themselves to construction heuristics, and instead will incorporate a local search after the route construction [49].

2.3 Local Search Heuristics

Local search heuristics take an input of a feasible solution, and slightly change the solution (while ensuring it remains feasible) to improve solution quality. As the initial solution typically results from a construction heuristic, these methods typically take more computation time. These have taken many different forms, but their defining trait is that the changes are small and localized [6, 32, 49]. Some heuristics focus on eliminating places that paths cross, some attempt to exchange a delivery in one route to another, and some allow for trading deliveries between routes, seeking to improve both simultaneously. In some cases, if a penalty is applied to infeasable solutions (penalizing for customers that are not serviced) a local search algorithm can be used to construct a solution from scratch, without

the need of a construction heuristic [32].

2.4 Metaheuristics and Adaptive Solution Methods

Metaheuristics are an expansive category that sometimes strays far from that of local search, though sometimes the distinction is less clear. Some notable metaheuristic methods include simulated annealing [8, 17, 21, 65], tabu search [61, 66], ant colony optimization [4,9,22,48], particle swarm optimization [13,31,37,45] and genetic and evolutionary algorithms [28, 46].

2.4.1 Simulated Annealing

"Simulated annealing" is a process that forms an analogy to the physical process of annealing. In physical annealing, a material is heated to a high temperature, and gradually cooled such that the material is allowed to settle into a low-energy state, typically characterized by high organization and strength properties [8].

Similarly in optimization, simulated annealing seeks to find solutions that are of high organization and strength to disturbance. The utility of the solution (which is to be maximized) can be thought of as an analogy to the internal energy of the physical material (which is to be minimized) [40].

As the state is perturbed slightly, this "energy" is measured, and the new system state is accepted if it is of lower energy. If it is of higher energy, it is accepted with a probability proportional to a parameter called temperature, which

is slowly reduced over the cycle of computation. The schedule by which the cooling occurs is a parameter that typically affects resultant algorithm performance to a high degree [17].

2.4.2 Tabu Search

"Tabu search" is a method that is essentially the same as a local search method, with the exception that it forbids the visitation of recently-visited solution variations. This serves to force the algorithm out of local minima, and encourages the exploration of other areas of the search space. When in discrete combinatorial problems, like the VRP, it can perform well [66], but continuous domains can cause problems with the definition of exactly what a forbidden region is; some area around a previously visited point must be forbidden, and the size of this area typically must be defined by a designer, or significant computation time is necessary [61].

Tabu searches, like most metaheuristics, have varying levels of complexity that are largely controlled by the designer. Lean tabu searches are the most popular and effective today, though lean approaches from the past have sometimes had significant oversights [16]. Many tabu search methods have a high number of used-controlled parameters, however, which can lead to a variety of problems, including a false reduction of search space size caused by experimentally choosing parameters on the same data as is used for testing [34].

2.4.3 Ant Optimization

"Ant optimization", is a type of optimization that is centralized around the concept of using stigmergy, wherein a series of solutions is developed using a "collective memory" that is common to all the solutions and some type of heuristic. Once these solutions are developed, the "collective memory" is biased toward the better solutions. Over the course of many iterations, this memory value will build up around the best of the available solutions, guiding the system toward a favorable solution [48].

The way that Ant Colony Optimization works in the VRP is as follows: an ant starts at the depot, and proceeds to a nearby customer with a probability based on both the amount of "pheremones" on that trail, and the inverse of the distance to that customer. The ant repeats this process until a vehicle tracing the ant's path would have no more capacity for deliveries, at which point it returns to the depot, and then proceeds by selecting another customer not yet visited. The ant leaves additional pheremones upon the trails that it traverses, for future ants. One ant completes its path, visiting all customers, before the next ant begins its path. The pheremones left behind decay with the number of ants, so that a pheremone left late in an ant's journey does not have more effect than one left early in the journey [5].

The Ant Colony System has been used as a method for decomposing multiobjective treatments of the VRP, such that each ant colony only has to consider a single objective as it lays pheremone trails [4,22]. By properly tuning the rates that

these pheremone trails are created and eliminated, this method attains solutions that are suitable on both multiple metrics simultaneously. Multiple independently-acting ant colonies have also been suggested as a way of alleviating some scaling issues experienced by Ant Optimzation [5].

2.4.4 Particle Swarm Optimization

"Particle Swarm Optimization" (PSO) is a technique in which a group of solutions is randomly generated, and then iteratively "move" throughout the space of parameters that describe the solutions, being "pulled" both toward their previous personal best position, as well as the current global best position. It was originally proposed by Kennedy in 1995 [31], and has been applied numerous times to the VRP [13, 37, 45, 53]. It bears the advantage of not needing to calculate a gradient of the function to minimize, and typically uses very simple mathematical computations.

It works by taking each solution (particle) within a population (swarm) and changing the parameters that define it to be closer to its best-discovered parameters, and then toward the global-best current set of parameters. By iterating this process many times, the solutions explore the parameter space, while being biased toward higher solution quality, leading to a steady improvement in the resultant solutions. In some implementations, the move toward the global-best is reduced to a move to the best solution within a local neighborhood.

Various techniques for smoothing the performance of this method over rough

objective functions exist, including the use of a "momentum" term, which causes a particle to maintain some inertia in the previous direction it was moving. Additionally, the tradeoff between self-reliance and social-reliance can be controlled through a pair of parameters that multiply the effect of each of these influences [64].

2.4.5 Evolutionary Algorithms

"Evolutionary algorithms" is the broadest category that we discuss here. An evolutionary algorithm is characterized by a population of candidate solutions (individuals). Each of these individuals carries a complete set of parameters necessary to describe a solution, and are initially generated randomly. After testing each of the individuals in a simulated environment and assigning a scalar fitness value to each individual, the algorithm proceeds by selecting those that will "survive", and those that will be eliminated. To replace the eliminated solutions, surviving individuals are replicated, and mutated slightly to produce new individuals that are nearby within the solution space [42, 43]. Through repeating this process over many generations, the population is biased toward areas of the solution space that result in higher quality system performance.

Evolutionary algorithms are criticized for being computationally expensive. This can be true compared to many of the other algorithms discussed here, but the computation expense can vary widely between different evolutionary algorithms [63].

The computational complexity of an evolutionary algorithm is directly re-

lated to the number of parameters required to describe an individual [63], so the choices made by the system designer in the use of an evolutionary algorithm are paramount. Intelligently decomposing the problem such that smaller sub-problems can be solved, resulting in favorable system behavior is one way of reducing computational complexity [42, 63]. Another way is by developing smaller computational units and combining them with an evolutionary method [23, 29]. One additional method for reducing the computation time of an evolutionary algorithm is the use of transfer learning, in which an individual or population is trained in a "source" environment and then applied to a "target" environment [25, 41].

Each of these methods seeks to simplify the external environment or the internal model of an agent in order to make the evolutionary computation less intensive. Two of these methods, transfer learning and coevolution, bear significantly on this work and are described in the following sections.

2.5 Transfer Learning

Given the complexity of the FCP formulation, any attempt at using a construction heuristic or local search heuristic would require the defining of new heuristics to address the additional extensions, especially the variable rate-of-travel. The metaheuristics show promise in being able to address these concerns, though the continuous action space associated with the variable rate-of-travel could lend problems to the implementation of some, like simulated annealing or ant colony optimization.

To this end, we identified evolutionary algorithms as one metaheuristic that showed promise. One additional benefit offered by evolutionary algorithms over the other metaheuristics was that, if designed properly, the solutions could generalize across problem instances.

A centralized evolutionary method that simultaneously controls all of the vehicles associated with an FCP instance would be very complex, with an extremely large solution space.

An adaptive, multiagent approach, on the other hand, could break the problem up such that local information could be used for agent decisions. In such an approach, an agent would sense information about its environment, reason based on that information, and take some action, seeking to maximize a system utility measure. By using many such agents, we create a multiagent setting which allows for an effectively decentralized approach with minimal communication between agents. Using adaptive agents can also allow the use of transfer learning (TL) to leverage experience gained in one problem instance to increase performance in another problem instance [11, 12].

Transfer learning takes various forms, but the unifying principle is that learning in a "source" domain that is somehow related to the "target" domain can boost performance in the target domain, even though the algorithm has never seen the target domain.

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 [59]. However, transfer learning can also be lever-

aged 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. This can either be done with a direct policy transfer [11], or through some explicit mapping [58].

An agent trained on source problems similar to the target problem will gain benefits in performance, training speed or trainability in the target domain, but agents trained on a random or unrelated task will not gain any performance benefits, and may in fact be hurt by the transfer [12, 41].

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 [25, 41]. 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. We address this need with respect to this work in Chapter 4. It is important to note that we do not explicitly create a mapping from our source task to our target task, and the policies do not need to be altered in any way in this work for the transfer to be successful.

To ease the search space requirements associated with an evolutionary algorithm, we elected to use transfer learning as a method to allow us to train first on a simple microcosm that models a small portion of the FCP. This also nicely accomodated our vehicle-centric multiagent approach; the domain that we developed was a single-agent implementation and is described more thoroughly in Chapter 4

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 [12,47]. These neural networks have been used in applications as complex and diverse as computer vision, HIV therapy screening, and coronary artery disease diagnosis [1, 7, 50].

2.6 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 [42]. 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 [42, 47]. This can be due to the agents learning to take a conservative strategy, being able to cooperate with a broader range of teammates [42,43]. However, in this work we do not seek to find the global optimum, as the system dynamics are extremely complex. Instead, we seek to find a solution that suitably solves the problem. Using biasing techniques addressed by Panait to boost performance toward the global optimum is a possible extension of this work.

Chapter 3 – Fleet Coordination Problem

> "The natural formation of the country is the soldier's best ally; but a power of estimating the adversary, of controlling the forces of victory, and of shrewdly calculating difficulties, dangers and distances, constitutes the test of a great general." - *Sun Tzu*

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 2 : FCP** *The Fleet Coordination Problem (FCP) consists of an allotted time* $T = \{t | 0 \leqslant t \leqslant T_{end}\}$*, 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* $0 \leqslant w_i \leqslant W_{max}$*, 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 $R_k \in [1, 100]$ 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 4.6).

There are a few key points to note in this formulation, which is also described in Algorithm 1. 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, 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 [54].

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:

$$F_{tot} = \sum_{k=1}^{nK} F_{elec,k} + F_{req,k} \tag{3.1}$$

where

$$F_{req,k} = \eta_i \alpha_1 \delta_k (1 + w_i \sin(S_k)) \tag{3.2}$$

$$F_{elec,k} = \eta_i \alpha_2 \delta_k R_{j,k}^2 + \eta_i \alpha_3 \delta_k R_{j,k} \tag{3.3}$$

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 [54]. 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 3.2 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 3.3 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) [54].

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 the state of the art. This model was chosen to limit computation time, while still lending a degree of realism to the problem domain [38].

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

$$T_{tot} = \sum_{k=1}^{nK} \frac{\delta_k}{R_{j,k}} \tag{3.4}$$

where T_{tot} is the total time it takes for the vehicle to travel from the two locations, λ_a and λ_b. This process is shown in Algorithm 2.

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 model of reality, and cannot incorporate all facets of the real-world problem.

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 : $v_{win} = \underset{i}{\text{argmax}}(\beta_{i,j})$
  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 (λ_a, λ_b)
  Decrease weight of v_{win} by package weight w_j
  Determine if package j was delivered within desired delivery window (Equation 4.2)
end for

Algorithm 2 Travel Algorithm

Given origin and destination locations (λ_a, λ_b)
Determine the length of each journey leg, δ_k
for $k = 1 \rightarrow journey\_legs$ **do**
  Select rate of travel $R_{(i,k)}$
  Calculate required and elective fuel spent (Equations 3.2 and 3.3)
  Calculate total fuel spent F_{tot} (Equation 3.1)
  Calculate time spent T_{tot} (Equation 3.4)
  Mark vehicle as "busy" for the time spent.
end for

Chapter 4 – Methods and Algorithms

> "The different measures suited to the nine varieties of ground; the expediency of aggressive or defensive tactics; and the fundamental laws of human nature: these are things that must most certainly be studied." - *Sun Tzu*

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 Chapter 3, and portrayed in Algorithm 1) is so complex, we decompose the agent responsibilities into two categories: (i) the movement of the vehicle from one location of another, and (ii) the decision of which vehicle 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 3.1-3.4).

The other agent responsibility is choosing which package pickup and delivery events will correspond to which vehicles. To solve this portion of the problem, we use a distributed, one-shot auction in which the highest bidder takes responsibility for travelling 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. This domain consists of a *single* driving agent that is placed in a location, and given a destination and a time limit to reach the location. This serves as the smallest possible microcosm of the FCP within which a driving agent may be trained. The source problem that we train the driver on is detailed as follows:

> **Definition 3 : 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 3.1), and arriving before a time $T_f \in T$.*

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 of the nK legs of the journey[1]) 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

[1] 10 was chosen as a representative value for nK. In reality, this calculation would be done in tandem with a GPS-based routing algorithm, and could take on any integer value.

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[2], 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 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:

$$U_{j,g} = -F_{tot} - H(g)L(j) \tag{4.1}$$

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 "handicap" assessed for arriving late and is a function of the

[2]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 3.3 for details.

generation, and $L(j)$ is calculated as:

$$L(j) = max(0, T_{arrive} - T_f) \tag{4.2}$$

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

In Equation 4.1, 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.

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.

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 ($\beta_{max} - \beta_{min} \leqslant 0.01$). 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, which very simply poses a question to a bidding agent:

of the vehicles available, placed in random locations around the package origin location, which of them should deliver the package to its destination? In this way, it serves as a microcosm for the FCP.

This domain is detailed below:

> **Definition 4 : 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:*
>
> $$\beta_{train} = 1 - k_1\delta_{(i,a)} - k_2 S_{(i,a)} - k_3\psi_j - k_4 w_v \tag{4.3}$$
>
> *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:

$$U_i = -|\beta_i - \beta_{train}| \tag{4.4}$$

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 4.3 is as follows: from the intent of formatting the problem in this fashion, we desire to have a high bid denote a vehicle that is a good candidate to make the delivery, and a low bid denote a vehicle that is a poor candidate. Utilizing the entire bidding range is desirable. A vehicle that is further from the pickup location is a worse choice for a delivery than one closer; a vehicle that travels uphill to get to a delivery is a worse choice than one travelling downhill; one that must rush to get to the pickup location on time is a worse choice than one that has ample time; and a heavier vehicle is a poorer choice than a lighter one. That is, in general, a vehicle that has to spend less fuel on a delivery is a better choice to make that delivery.

We chose to model this with a simple linear equation because it was the simplest possible case that provided all of these desired traits. Note that during the evolutionary process in the target domain, the bidders are free to stray from this initial bidding bias, if it increases their fitness measure.

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.

$$\mathbb{L}_i = \sum_{j=1}^{nP} I_i(j)[-F_{(tot,j)} - HL(j) + \phi] \tag{4.5}$$

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 4.2, 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. When set at a value that is not near these two extremes, however, the system performance is not too sensitive to this parameter. In a more detailed future implementation, this could be easily set to correspond to the value of the delivery to the carrier company, however, in this implementation, by keeping the delivery reward constant we are able to leave it out of the global reward calculation.

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

$$\mathbb{G} = \sum_{i=1}^{nV} [\mathbb{L}_i] - (nP)\phi \tag{4.6}$$

At the system level we are concerned with the sum total of fuel spent and delivering packages on time; 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 to 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 4.5, 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:

$$f(x) = \begin{cases} 0 & : x < -2 \\ 0.25x + 0.5 & : -2 \leqslant x \leqslant 2 \\ 1 & : x > 2 \end{cases} \tag{4.7}$$

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.

4.6 Verification of Optimality in Simple Conditions

Finally, we wished to show that our approach is both (i) able to discover the optimal policy in a case where we can verify the optimal policy, and (ii) that it scales well with the complexity of the problem instance.

To show the former, we create a problem instance where two clusters of cities are separated by a large distance, and each has one truck to service it. This allows us to directly calculate the optimum fuel expenditures, and compare this with the performance of our solution technique outlined above.

To show that our approach scales better than an alternative metaheuristic, we implemented two types of simulated annealing algorithms:

- Centralized Simulated Annealing: In this algorithm, a centralized controller keeps a list of assignments that pairs each package uniquely with the vehicle responsible for it, and another centralized lookup table that determines the rate of travel that any vehicle should take based on a discretization of its state.

- Distributed Simulated Annealing: In this, each vehicle keeps a list of package priorities, where each package is given a unique priority in the range [1,nP],

and in a manner similar to the one-shot auction used in our neural-network solution methodology, the vehicle with the highest priority takes responsibility for the package. Additionally, each vehicle keeps a lookup table that determines the rate of travel the vehicle will take based on a discretization of its state.

We show in Section **??** that both our method and these metaheuristics discover the optimal policy in the simple case, and that in the more complex FCP instance used for the bulk of our results, we attain better performance with the coevolving populations of neural networks and transfer learning, with a reduced number of parameters.

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 β_i $\forall$ agents i (Equation 4.3, 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 4.1, 4.4, 4.5 or 4.6)
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 ± 0.05
end if
end for

end for

Chapter 5 – Results and Discussion

> "Thus it is that in war the victorious strategist only seeks battle after the victory has been won, whereas he who is destined to defeat first fights and afterwards looks for victory." - *Sun Tzu*

In this work, we compare the results of various solution strategies 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 bidder traning, we used values of $\{k_1, k_2, k_3, k_4\} = \{0.6, 0.1, 0.1, 0.2\}$. These values were chosen simply to represent that, while the other values hold some importance, the most important consideration in deciding on delivery rights is physical proximity (one would not re-route a truck in New York for a pickup in Los Angeles), which corresponds to k_1. We created a time scale 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 tested, 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 *unit-less* quantity that represents a combination of the fleet's fuel consumption and on-time performance (Equation 4.6). Error is reported as the standard deviation of the mean,[1] and in many cases is smaller than the symbols used to plot. To validate our distributed approach to the novel FCP presented in Chapter 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 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. We show results in the following scenarios:

1. No transfer learning with fitness calculated through local and global utilities (Section 5.1)
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. Impact of driver training length (Section 5.5)
6. Impact of simulated traffic (Section 5.6)
7. Simulated annealing and scaling (Section 5.7)

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

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 Chapter 4. Figure 5.1 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 4.6).

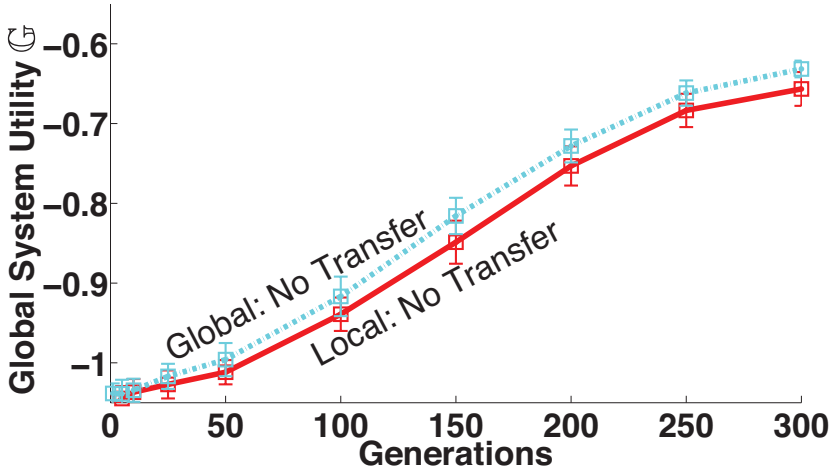

Figure 5.1: Comparison of agents evolved in the FCP using as their fitness evaluation: $\mathbb{G}$, the global system utility (upper, dotted line); $\mathbb{L}_i$ (lower, solid line). Note that in 300 generations, both fitness calculations lead to improvement in system performance.

5.2 Full Transfer Learning

We note that the full transfer learning case shown in Figure 5.2 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.

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 5.3 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

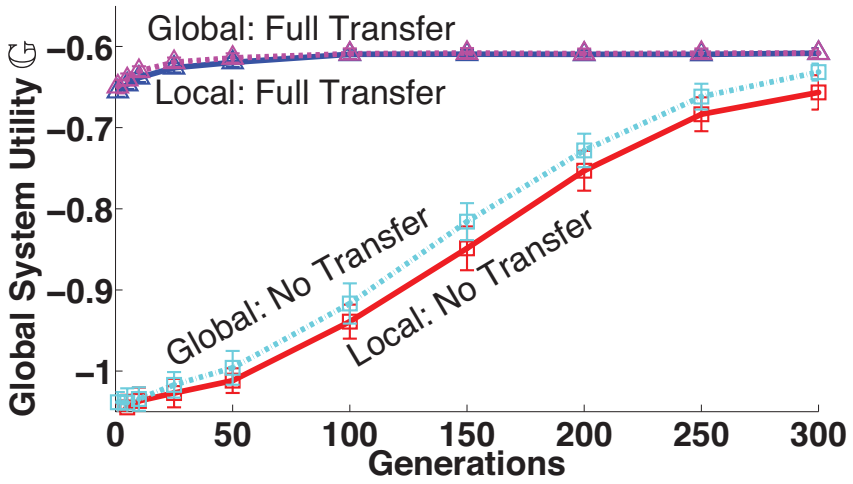

Figure 5.2: Performance in the FCP of agents coevolved solely in the FCP (lower lines with square markers) against agents evolved in separate source domains transferred into the FCP (upper lines with triangle markers). Note that the two full-transfer learning cases overlap significantly, and the boost 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.

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:

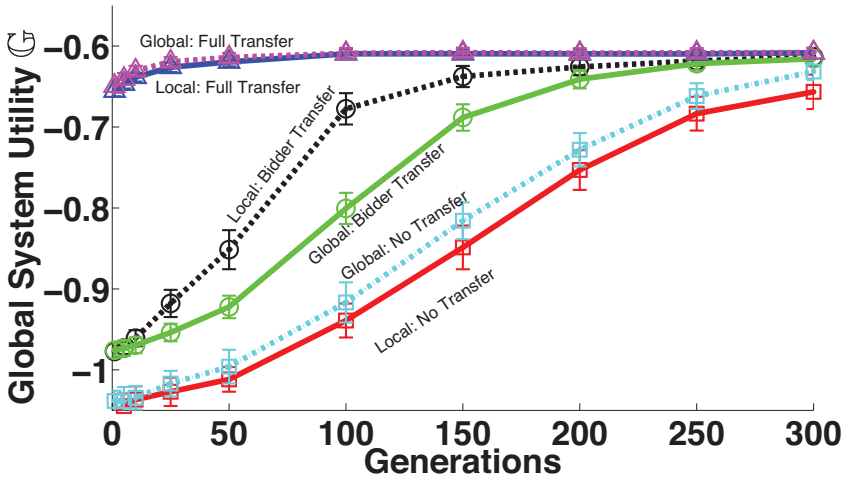

Figure 5.3: Experimental results in the FCP using three forms of transfer learning: no transfer (bottom, blue and red), full transfer (top, triangles), and bidder-only transfer learning (middle, green and black) which could correspond to an information loss scenario. Note that the full transfer learning cases overlap each other.

in the tested instances of the FCP, the drivers are capable of wasting far more fuel than poorly assigned bids, due to the quadratic portion of fuel cost calculation. These results were omitted from Figure 5.3 for readability. In higher package-congestion cases, 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

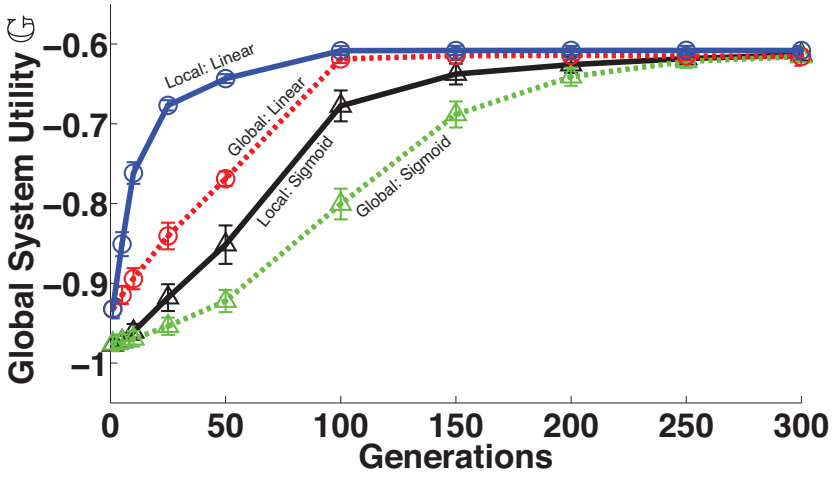

Figure 5.4: Experimental results in the FCP using bidder transfer learning and coevolving on the global system utility $\mathbb{G}$ or local agent utility $\mathbb{L}_i$. Because of the relatively simple problem instance, a neural network using a linear function as an approximation of the sigmoid itself actually outperformed the full-accuracy neural network.

the same slope as the sigmoid at zero, seen in Equation 4.7.

In Figure 5.4, 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. This is potentially due to the simpler mapping between inputs and outputs (thus being easier to learn), or that it is easier for the linear network to attain outputs that are closer to the extremes of zero and one respectively, possibly resulting in a simpler representation that still allows an ability to choose minimum or maximum speeds, in appropriate situations. 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.

5.5 Impact of Driver Training Time

Another matter of interest within this work is the impact that the training time in the driver source domain has on the initial performance in the FCP. We note that in the results shown thus far (Sections 5.1–5.3), there is a clear advantage to training in the driver source domain, but the question then becomes "How much is enough?"

To answer this question we conducted another series of experiments, wherein we trained the driver in the driver source domain (Section 4.2) for between 0 and 100 generations, and the bidder in the bidder source equation (Section 4.3) for 100 generations in each case. We then allowed the partially-trained driver and trained bidder to work together in the first generation of the FCP, and recorded the results, which are reported in Figure 5.5.

Benefits in initial performance are greater-than-linear for the first 30 generations, before diminishing returns causes the performance increase to taper off. After 70 training generations in the driver source domain, the initial performance

in the FCP is as high as was seen during these trials. The initial performance decrease seen after 70 generations could be a artifact of a phenomenon known as "overtraining" in which a neural network trained on similar data for too long may lose generalization power. It could also be due to the fact that as the drivers learn to be more and more efficient with their individual fuel expenditures, they take longer to complete deliveries, possibly causing another vehicle to take a longer route to complete a delivery that they might otherwise be available to conduct.

Even through training the driver agents on the DSD for 100 generations, the

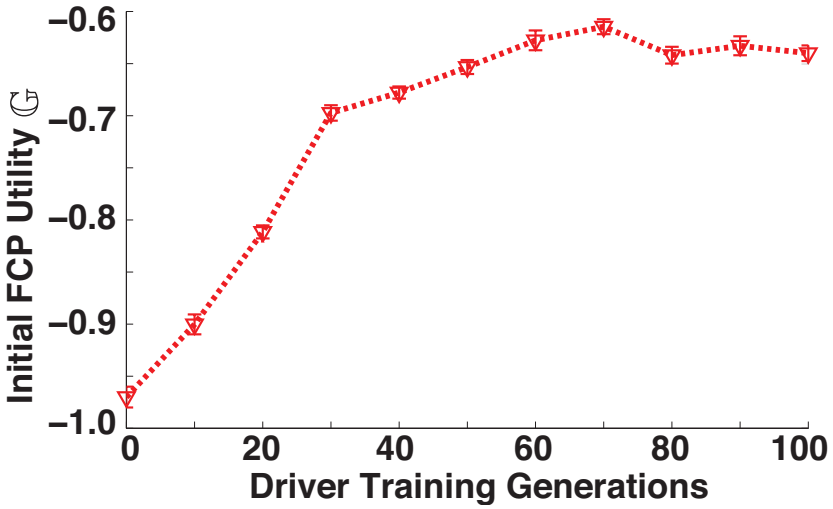

Figure 5.5: Initial performance in the FCP as a function of training generations allowed in the Driver Source Domain. In each case the bidder was trained for 100 generations on the Bidder Source Equation before the initial FCP performance was measured. The performance gain increases steadily for the first 70 generations, after which the benefits of transfer learning have been realized. After this, there is a small dip in performance, possibly caused by the neural network agents overtraining on the source domain. It is extremely important to note that training generations in this figure represent a very small fraction of the computation time required for a training generation in any other figure (the scales differ).

computation time is an order of magnitude less than training for 100 additional generations on the FCP, due to the simplicity of the DSD problem formulation; computation time for a single generation in the DSD is about 5% of that in the FCP, so training first in the DSD is extremely beneficial for total training computation time.

5.6 Simulated Traffic

Finally, we wish to examine the impact that an imperfect roadway system has on our algorithms. To simulate traffic and congestion on the roadways, for each leg of a journey we simply draw a random number ξ from a one-sided (positive) triangular distribution, ranging from $[0, \Xi]$. This ξ acts on the vehicles in two ways: first, it slows down the rate of travel ($R_{i,k} \leftarrow (1 - \xi) * R_{i,k}$), and second, it increases the fuel consumption ($F_{tot} \leftarrow (1 + \xi) * F_{tot}$).

Figure 5.6 shows that as the average traffic increases, system performance both degrades, and becomes more volatile. This is to be expected with such noise added to the system. It is important to note, however, that the agents allowed use transfer learning perform better in the presence of this increased noise than the agents that are not allowed to train first on the simple problem domains.

This problem formulation is difficult for agents to solve; agents will take thousands of actions before being assigned a fitness evaluation, and the actual effect of their actions is not deterministic. This adds learnability problems onto the already-noisy global fitness evaluation. The agents without transfer learning are

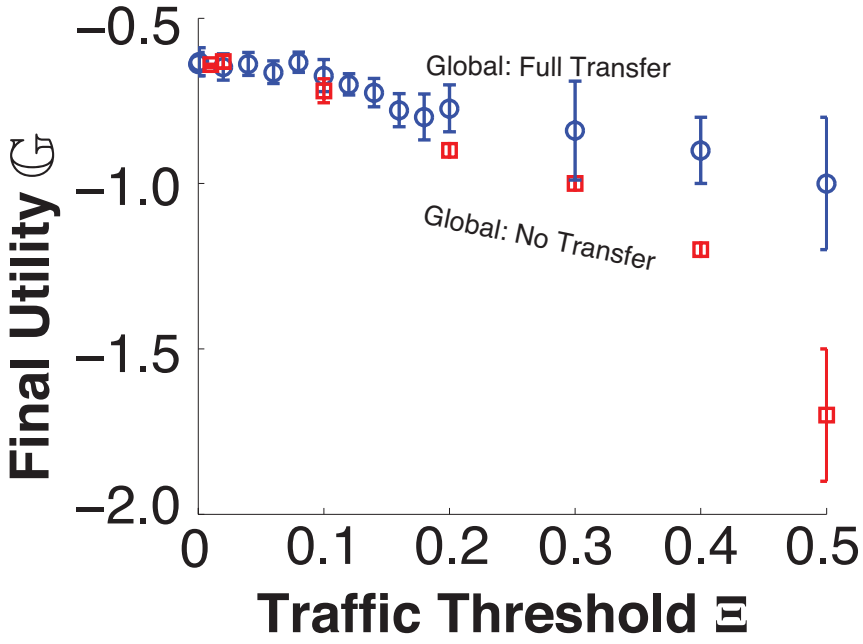

Figure 5.6: Converged FCP performance after 300 generations of full transfer learning (top blue circles) and no transfer learning (bottom red squares) as a function of traffic congestion, over 30 statistical runs. This congestion was drawn from a one-sided triangular distribution with range $[0, \Xi]$, and serves to simultaneously increase fuel consumption and decrease rate of travel. Agents using transfer learning see benefits as the congestion increases.

initially being evaluated for survival based on random actions taken in a noisy environment, which could lead to the culling of policies that might otherwise prove promising. The agents that used transfer learning, however, start with policies that significanlty outperform random. This means the evolutionary algorithm does not have the additional noise added by the randomly seeded actions on top of the noise created by the traffic. In short, the transfer learning allows for a clean enough initial signal that the algorithm is able to learn through the noise introduced by the

traffic, as long as the traffic distribution mean remains low enough.

5.7 Validation vs. Optimal Solution and Simulated Annealing

Finally, to validate our approach to breaking the problem down in this particular manner and addressing it with our solution strategy, we chose to look at a metaheuristic under two cases of the FCP for comparison. First, we attempted to apply two simulated annealing strategies (one centralized, one distributed, outlined in section 4.6) to the full FCP, to compare against our results. Because this works on a time scale that is different from generations, we do not show the learning graphs, and instead summarize the converged results in Table 5.1.

We attempted to tune the parameters appropriately to the problem, and eventually found that the best performance actually occurred by choosing only a single

Name	Parameters	Mean Converged $\mathbb{G}$
Full Transfer: Local	1220	-0.6088
Full Transfer: Global	1220	-0.6090
Linear Approximation: Global	1220	-0.6093
Linear Approximation: Local	1220	-0.6098
Bidder Transfer: Local	1220	-0.6166
Bidder Transfer: Global	1220	-0.6167
No Transfer: Local	1220	-0.6579
No Transfer: Global	1220	-0.6316
Centralized Simulated Annealing	1001	-0.8243
Distributed Simulated Annealing	10010	-0.8616

Table 5.1: Comparison in the full FCP of parameters required for each type of solution strategy in the as well as the converged solution quality achieved by each method.

speed at which each vehicle will travel. Beyond this, the algorithm ran into difficulties with changing values that were not used in a particular experimental iteration, that changed results in future iterations — that is, though a particular state might not be visited in an experimental iteration, the changes to the lookup table that are involved with that state might be saved to a different value. This, along with determining the correct cooling schedule, posed significant difficulties in gaining any traction in the problem. The best performance we found over a number of trials involved the single speed value for all states, removing the possibility of un-visited state information from changing.

Additionally, to verify that our solution strategy is providing acceptable results, we look at a simplified case of the FCP, compared to the one used for all other experiments.

In this case, the fleet consists of 2 trucks serving 4 customer locations over a course of 10 package deliveries. These deliveries are spaced out temporally so that the ideal solution is to travel at the minimum allowable speed. The customer locations are initialized spatially in two pairs; travelling from one city to its paired city requires a minimum fuel expenditure of 1, while travelling between the pairs requires an expenditure of 100. Each package's pickup and delivery points are opposite cities within a pair, at all points. The two trucks are initialized at cities in opposite pairs, at the site of the first pickup location, and the package origin city swaps for each new package.

Thus, it becomes immediately obvious that the correct response is for each truck to remain within its own pair, conducting all of these package deliveries

at the minimum rate of travel. This will lead to an exact solution of nP fuel expenditures, and no late package deliveries.

These results show that both our proposed solution method utilizing distributed neural network agents and transfer learning, as well as the simulated annealing approach, discover a near-optimal solution in a simple case in which we can calculate the optimal. However, as shown in Table 5.1, simulated annealing begins to break down as hte problem complexity increases, while our solution techniques remain robust to the more complex problem. We do not make any claims of optimality in the more complex instance, and merely claim that our results are favorable.

It is also important to note that we attain better performance with more parameters due to the nature of our solution strategy: instead of attempting to directly assign the packages to different vehicles, we are using a function approximator. It is possible that a less complex neural network could attain similar results.

Name	Parameters	Mean Converged Fuel Costs
Exact	n/a	10
Full Transfer: Global	244	10.0005
Centralized Simulated Annealing	11	10.0000
Distributed Simulated Annealing	21	10.0001

Table 5.2: Comparison in a very simplified FCP environment of the parameters required by each solution method and final converged fuel costs for each solution. Note that in this simplified case, all methods converge to very near the optimal solution.

Chapter 6 – Conclusion

> "The clever combatant looks to the effect of combined energy, and does not require too much from individuals."
> - *Sun Tzu*

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 direct 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 the properties of an FCP instance that cannot be successfully learned from scratch, but which is still solvable using transfer learning.

It would be interesting to delve further into the way that traffic is incorporated in this work: using time-of-day-dependent traffic levels, representing rush hour, or having higher mean traffic or variabilities in urban areas offer a number of interesting possibilities for expansion.

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.

Bibliography

[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. *European Conference on Computer Vision*, 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] Benjamin Baran and Matilde Schaerer. A multiobjective ant colony system for vehicle routing problem with time windows. *International Association of Science and Technology for Development International Conference on Applied Informatics*, pages 97–102, February 2003.

[5] John E. Bell and Patrick R. McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18(1):41 – 48, 2004.

[6] Dimitris J. Bertsimas. A vehicle routing problem with stochastic demand. *Operations Research*, 40(3):pp. 574–585, 1992.

[7] Steffen Bickel, Jasmina Bogojeska, Thomas Lengauer, and Tobias Scheffer. Multi-task learning for hiv therapy screening. *International Conference on Machine Learning*, pages 56–63, 2008.

[8] Alex Van Breedam. Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research*, 86(3):480 – 490, 1995.

[9] Bernd Bullnheimer, Richard F. Hartl, and Christine Strauss. Applying the ant system to the vehicle routing problem. *International Conference on Metaheuristics*, 1997.

[10] Lawrence D. Burns, Randolph W. Hall, Dennis E. Blumenfeld, and Carlos F. Daganzo. Distribution strategies that minimize transportation and inventory costs. *Operations Research*, 33(3):469–490, May 1985.

[11] L. Cardamone, D. Loiacono, and P.L. Lanzi. Learning to drive in the open racing car simulator using online neuroevolution. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(3):176 –190, Sept. 2010.

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

[13] Ai-ling Chen, Gen-ke Yang, and Zhi-ming Wu. Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. *Journal of Zhejiang University - Science A*, 7:607–614, 2006. 10.1631/jzus.2006.A0607.

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

[15] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):pp. 568–581, 1964.

[16] J-F Cordeau, M Gendreau, G Laporte, J-Y Potvin, and F Semet. A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, 53:512–522, 2002.

[17] Z.J. Czech and P. Czarnas. Parallel simulated annealing for the vehicle routing problem with time windows. In *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on*, pages 376 –383, 2002.

[18] 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.

[19] 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.

[20] 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.

[21] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106:491–511, 2006. 10.1007/s10107-005-0644-x.

[22] Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi. Macs-vrptw: A multiple colony system for vehicle routing problems with time windows. In *New Ideas in Optimization*, pages 63–76. McGraw-Hill, 1999.

[23] Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Efficient non-linear control through neuroevolution. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 654–662. Springer Berlin / Heidelberg, 2006.

[24] 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.

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

[26] Ali Haghani and Soojung Jung. A dynamic vehicle routing problem with time-dependent travel times. *Computers and Operations Research*, 32(11):2959 – 2986, 2005.

[27] 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.

[28] Jorg Homberger and Hermann Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR*, 37(3):297–318, 1999.

[29] Gomez F. J. Solving non-markovian control tasks with neuro-evolution. *Proceedings of IJCAI-99*, 1999.

[30] 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.

[31] James Kennedy and Russell Eberhart. Particle swarm optimization. *Proceedings IEEE Inter- national Conference on Neural Networks*, pages 1942–1948, 1995.

[32] Philip Kilby, Patrick Prosser, and Paul Shaw. *Guided local search for the vehicle routing problem*, volume 1, pages 21–24. Citeseer, 1997.

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

[34] Gilbert Laporte, Jean-Yves Potvin Michel Gendreau, and Frédéric Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 2000.

[35] 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.

[36] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):pp. 498–516, 1973.

[37] Babak Farhang Moghadam and Seyed Mohammad Seyedhosseini. A particle swarm approach to solve vehicle routing problem with uncertain demand: A drug distribution case study. *International Journal of Industrial Engineering Computations*, pages 55–66, 2010.

[38] Daimler Trucks NA. Personal communication.

[39] 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.

[40] Ping-Feng Pai and Wei-Chiang Hong. Support vector machines with simulated annealing algorithms in electricity load forecasting. *Energy Conversion and Management*, 46(17):2669 – 2688, 2005.

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

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

[43] 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.

[44] Yang-Byung Park and Sung-Hun Song. Vehicle scheduling problems with time-varying speed. *Computers in Industrial Engineering*, 33(3-4):853–856, 1997.

[45] Yang Peng and Ye mei Qian. A particle swarm optimization to vehicle routing problem with fuzzy demands. *Journal of Convergence Information Technology*, 5(6), August 2010.

[46] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31(12):1985 – 2002, 2004.

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

[48] Marc Reimann, Karl Doerner, and Richard F Hartl. D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers and Operations Research*, 31(4):563 – 591, 2004.

[49] Robert A. Russell. Hybrid heuristics for the vehicle routing problem with time windows. *Transportation Science*, 29(2):156–166, 1995.

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

[51] 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.

[52] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):pp. 254–265, 1987.

[53] Pandhapon Sombuntham and Voratas Kachitvichayanukul. A particle swarm optimization algorithm for multi-depot vehicle routing problem with pickup and delivery requests. *Lecture Notes in Engineering and Computer Science*, 2182(1), 2010.

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

[55] 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.

[56] Matthew E. Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 879–886, New York, NY, USA, 2007. ACM.

[57] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, December 2009.

[58] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Transfer learning for policy search methods. *International Conference on Machine Learning*, 2006.

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

[60] Lisa Torrey, Jude Shavlik, Trevor Walker, and Richard Maclin. Skill acquisition via transfer learning and advice taking. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 425–436. Springer Berlin / Heidelberg, 2006.

[61] Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003.

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

[63] Vinod K. Valsalam and Risto Miikkulainen. Modular neuroevolution for multilegged locomotion. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 265–272, New York, NY, USA, 2008. ACM.

[64] Wanliang Wang, Bin Wu, Yanwei Zhao, and Dingzhong Feng. Particle swarm optimization for open vehicle routing problem. In De-Shuang Huang, Kang Li, and George Irwin, editors, *Computational Intelligence*, volume 4114 of *Lecture Notes in Computer Science*, pages 999–1007. Springer Berlin / Heidelberg, 2006.

[65] Marcin Woch and Piotr Lebkowski. Sequential simulated annealing for the vehicle routing problem with time windows. *Decision Making in Manufacturing Sciences*, 3(1-2):87–100, 2009.

[66] Jiefeng Xu and James P. Kelly. A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Logistics*, 30:379–393, 1996.